Hidden Markov models

Natural language modelling & interfaces mini-project
March 11, 2018

Bram Otten & Lennart Beekhuis
Universiteit van Amsterdam

Abstract

We tested different emission and transition parameters for a hidden Markov model. To determine the best performing parameters, we tested our models using logarithmic perplexity and the Viterbi algorithm. We found three parameter configurations, which give the best perplexity, the best accuracy and an average performance on both perplexity and accuracy. We concluded that the value of the emission parameter is much more important than the value of the transition parameter, and that logarithmic complexity and accuracy decrease as we increase the emission parameter.

1 Introduction

In this project part-of-speech (POS) tagging is performed. This is the process of marking tokens from a sentence (individual words, but also comma's, punctuation marks, et cetera) with their function. This is done because, for example, the same words can have a different meaning depending on the way they are used. 'Being' can be a verb but also a noun. POS tagging is a disambiguation task, deriving the meaning of ambiguous tokens from context and previous 'experience' (data). This is a task humans are not much better at than machines using the approach that will be described.

The principle behind this approach is based on the assumption that parts of speech follow other parts of speech with a different likelihood, and that certain tokens are more likely to be a certain part of speech than another. These are respectively the transition and emission parameters. With a bi-gram assumption for the transition and a uni-gram assumption for the emission, this gives us the intuitive most likely tag sequence:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(t_i|t_{i-1})\, P(w_i|t_i)$$

Where t means a tag, w means a token (usually a word), the transition parameter $\lambda_{t_i}^{t_{i-1}} = P(t_i|t_{i-1})$ and emission parameter $\theta_{w_i}^{t_i} = P(w_i|t_i)$.

2 Model

Markers for unknown words and beginnings and endings of sentences are included during the modelling as needed, but we mainly use NLTK's universal tagset, containing the following tags:

- NOUN (nouns);
- VERB (verbs);
- ADJ (adjectives);
- ADV (adverbs);
- PRON (pronouns);
- DET (determiners and articles);
- ADP (prepositions and postpositions);
- NUM (numerals);
- CONJ (conjunctions);
- PRT (particles);
- . (punctuation marks);
- X (a catch-all for other categories such as abbreviations or foreign words).

The labeled dataset we will use is the English Penn treebank corpus will. This treebank contains many more tags than are in the universal set, these will be 'cast' to one of the universal ones. The treebank consists of 3914 hand-labelled sentences.

We will use a hidden Markov model (HMM) to predict the most likely tags for a sentence. Our HMM will use the assumptions from the introduction, and probabilities per state-change will only depend on the previous state and current state. (Which makes it quite similar to a single Markov chain.) In this case, that means the tag a word gets depends only on the one previous tag and the current word. The best paths to get to a certain tag for a certain word are used, which makes this algorithm much more efficient than choosing the most likely from every possible tag sequence for a sentence.

The implementation requires a few tricks. One is smoothing to prevent probabilities of 0. The emission parameter gives words that have never been seen a probability of:

$$\frac{\text{'emission alpha'}}{\text{number of words for that tag}}$$

The same goes for the transition parameter, every tag is defined to follow every other tag at least 'transition alpha' times. The value of these alpha's will be tweaked in the next sections. Another trick is that rather than taking the product of probabilities, we add up the natural logarithms of all probabilities. This operation is harmless because it can be reversed by raising Euler's number to the power of the log probability, yet it does alleviate the problem computers have with very small numbers.

3 Parameter estimation

To estimate the 'emission alpha' and 'transition alpha' parameters, we use two methods to test the quality of our model: (logarithmic) perplexity and accuracy compared to a gold standard which is hand-tagged by humans.

The perplexity of a model is the inverse probability of the test set, normalized by the number of words. We take the marginal probability of a sentence and then raise it to the power of -1 divided by the number of words in the sentence to obtain the perplexity. For the logarithmic perplexity, which we will use, the formula is slightly different:

$$\log \mathrm{PP}(\mathcal{T}) = -\frac{1}{t}\sum_{k=1}^{m} \log P_{S|N}(\langle x_1^{(k)}, \ldots, x_{n_k}^{(k)}\rangle | n_k; \boldsymbol{\theta})$$

A more intuitive way of thinking about perplexity is that it is the weighted average branching factor of the model (Jurafsky, 2000). A lower perplexity means there are less states that can follow an initial state. Out of these fewer possibilities, we are more likely to pick the right follow-up state. This is also the reason for which datasets with a greater number of words will tend to have greater perplexities.

Accuracy is measured by a comparison with hand-made tag sequences. We will consider these a gold standard even though humans may disagree among each other about the correct tag for a certain token. In any case, the comparison, and thus the accuracy, is simply:

$$\frac{\text{number of correctly tagged tokens}}{\text{total number of tokens}}$$

4 Experiments

To find the best parameters, we split the Penn treebank corpus up into a training set, a development set and a test set. They contain, respectively, 3000, 100 and 814 sentences. To test certain parameters, we first train a HMM using the training set and the parameters to be tested. Then, we compute the log perplexity of the trained model on another set (either the development set, or the test set). Finally, we compute the performance of the Viterbi algorithm on the other set.

Terminology may get a little confusing around now, but alpha stands for the emission parameter (also called 'emission alpha' above), beta for transition parameter ('transition alpha'). In any case, the results are shown in the following table

Results on development set (trained with training set):

alpha	beta	log pp	acc
0.01	0.01	53.449	0.892
0.01	0.1	47.331	0.887
0.01	1	41.961	0.856
0.01	10	41.162	0.768
0.1	0.01	53.445	0.892
0.1	0.1	47.33	0.886
0.1	1	41.96	0.856
0.1	10	41.162	0.768
1	0.01	53.414	0.892
1	0.1	47.317	0.886
1	1	41.957	0.856
1	10	41.161	0.768
10	0.01	53.127	0.891
10	0.1	47.194	0.887
10	1	41.922	0.862
10	10	41.156	0.769

Based on these results, we decided to further explore two parameter configurations:

- (alpha, beta) = (10, 10), because this configuration resulted in the lowest perplexity;
- (alpha, beta) = (0.01, 0.01), because this configuration resulted in the highest accuracy.

Note that (alpha, beta)'s of (0.1, 0.01) and (1.0, 0.01) have the same accuracy, but our chosen configuration has the lowest perplexity of these three options.

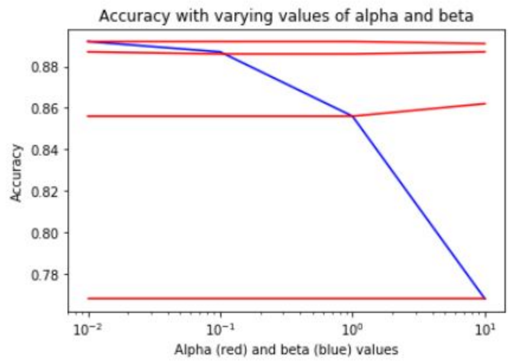
We also wanted to further test a configuration that had an average performance on both perplexity and accuracy. We chose (alpha, beta) = (10.0, 1.0) as our average performing configuration.

Results of our chosen parameter configurations on the test set (trained with training set again):

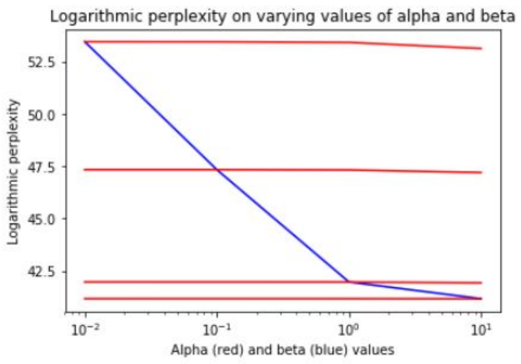
alpha	beta	log pp	acc
10.0	10.0	48.2	0.78
10.0	1.0	48.9	0.87
0.01	0.01	62.1	0.89

5 Conclusion

From our results, we can conclude a few things. First of all, the beta (emission) parameter is the more important parameter of the two. Varying beta changes the result much more than varying alpha.



Second, as we increase beta, accuracy goes down, and logarithmic perplexity goes down. It seems likely there is a point where the decrease of the logarithmic perplexity decreases, but we didn't have enough time to further investigate if this is, in fact, true.



In conclusion: The parameter beta (emission of word given tag) is much more important than the parameter alpha (transition of tags). A low value for beta gives the best result for accuracy, and a higher value for beta gives a better logarithmic perplexity.

If we had more time to optimize parameters, we would try alpha values between 10 and 100, to see if a high alpha value has more effect on the results. For beta, we would try values between 1 and 10, as we got the best overall results with these beta values.

References

Jurafsky, D. (2000). Speech and language processing: An introduction to natural language processing. *Computational linguistics, and speech recognition*.