Research Project

MARKOV POS TAGGER

Universiteit van Amsterdam

Yorick de Boer (10786015)
Amsterdam, University of Amsterdam March 7, 2016

Contents

Abstract

In this project we developed a POS tagger using the Viterbi algorithm in Python. To increase accuracy the bigram language model and word-tag pair emission model were smoothed using good turing smoothing. The program is able to determine the accuracy of the tagger by using a validated test set. Without smoothing the POS tagger has an accuracy of 32 percent and with GT smoothing the accuracy goes up to 35 percent.

Introduction

For this assignment we implemented a Markov POS-tagger. A POS tagger (Part-of-speech tagger) is a program that will tag each word grammatically. This particular POS tagger will choose a tag sequence for each sentence with the highest probability.

The reason it is called a Markov POS-tagger is because the program is using a Hidden Markov Model (HMM) to compute the probabilities. A HMM model uses the tag of the previous word to determine the best probability of the current tag, in this way the model is context bearing and therefore more reliable. All the possible tags for each word are represented as states. To determine the best tag sequence the optimal path between these states must be found. Using the viterbi algorithm we can determine this optimal path.

We were given a test and trainings corpora. Using the trainings corpora we will make a language model. In the language model we will compute the probability of a word with its corresponding tag will follow the previous word with the corresponding tag. These pairs are called bigrams. So now the probabilities of all bigrams of the trainings corpora are known, using this language model and the viterbi algorithm the best tag sequence for all the sentences in the test corpora can be determined.

Because the trainings and test corpora are both sections of the Wall Street Journal the hypothesis is that the language model is a good model to determine the tag sequences of the sentences in the test corpora with reservations that all the code is working well.

Method

We implemented a Markov POS-tagger in two steps: (1) a training step where we can estimate the probabilities using a training corpus and (2) a tagging step where a sentence is given as input and an the tag sequence with the highest probability is given as output. In step (1) there is also an option added for the user to smooth the data. After these two steps we evaluated the program by computing the accuracy. This assignment is made in the programming language Python.

Transition model and Emission model

In the previous assignments we were asked to make language model of bigrams given a corpora, we can use the code for this assignment to make a language model for this assignment. To make a language model we need a transition model and a emission model. Our code to make the transition model is made using the following equation, note that these are the bigrams of tags(t_i *and* t_{i-1}) and we used bigrams because of the markov assumption:

$$\prod_{i=1}^{n} P(t_i|t_{i-1}) = \frac{Count(t_i, t_i - 1)}{Count(t_i)} \tag{1}$$

t_i = *the tag at index* i, n *is length of words in corpora*

We could use the code from the previous assignments to determine this probability. All the counts to make this language model are taken from the trainings corpora.

For the code of the emission model we used the following equation:

$$\prod_{i=1}^{n} P(w_i|t_i) = \frac{Count(w_i, t_i)}{Count(t_i)} \tag{2}$$

w_i = *word at index* i, t_i = *the tag at index* i, n *is length of words in corpora*

These counts are also taken from the trainings corpora. The emission model and transition model are both stored in a python dictionary. Now these probabilities are known we could continue to step 2.

The viterbi alogrithm

Now the transition model and the emission model are known we can use the viterbi algorithm to implement a tagging step where a sentence is given as input and the tag sequence with the highest probability for that sentence is given as output. The viterbi uses the markov assumption to determine the optimal path in a Hidden Markov Model. To use the viterbi algorithm we must make a viterbi matrix. To fill this matrix we used the following equation:

$$v(j, t) = max_{i=1}^{N} v(i, t-1) a_{ij} b_j(O_t) \tag{3}$$

$v(j,t)$= *viterbi matrix v at index (j,t),* a_{ij}= *transition matrix a at index (i,j),* $b_j(O_t)$ = *emission matrix b at index j for output* O_t

Our code is based on this equation where a is the transition model and b the emission model. Our viterbi method will get return the most likely tag sequence given a sentence. We ran the viterbi algorithm on all sentences in the test set so all the words get tagged.

Evaluation

Because the POS tagger will probably have many zero counts we wanted the program to be able to do smoothing. We used Good Turing smoothing, with the standard Good Turing formula to compute the new count r:

$$r^* = \frac{(r+1)\frac{n_r+1}{n_r} - r(k+1)\frac{(n_{k+1})}{n_1}}{1-(k+1)\frac{n_{k+1}}{n_1}} \tag{4}$$

We only made use of Good Turing smoothing on the language model for words that appeared no more than 4 times(k¡4). For the lexical model we used Good Turing smoothing for frequencies of 0 and 1, using this formula:

$$O^* = \frac{n_1(t)}{2n_o}1^* = \frac{1}{2} \tag{5}$$

For computing the accuracy we used the following formula:

$$Accuracy(tagger) = \frac{(numberofwordscorrectlytaggedbytagger)}{N} \tag{6}$$

Where N is the length of the test corpus. We first removed the tags from the test corpora and ran our own tagger to determine the tags of the words in the test corpus. And then we compared our own results with the original test set to compute the accuracy.

Results

Smoothing	**Accuracy percentage**	**Difference percentage**
non	32.96	92.01
Good Turing	35.91	92.66

Table 1. Accuracy results

For list of generated POS tags see the appendix.

Discussion and Conclusion

The obtained accuracy results of the completely correct generated sentences are not very high at 35 percent. We expected these to be higher. This expectation comes from the fact that both the training and test corpus come from the same same source, the Wall Street Journal corpus. To determine how much the generated sentences differ from the validated sentence the percentage of difference was calculated for tagged sentence. It can be seen that the sentences are almost completely correct with a 92 percent on average. Most of the time this means that only one word in a sentence is wrongly generated.

The small difference in accuracy between the smoothed and the non smoothed models is most likely caused by an error in the code, the accuracies of the smoothed model should have been significantly higher than the accuracy of the non smoothed model.

Appendix

Run command

```
usage: a1step4.py [-h] [-smoothing SMOOTHING] [-train_set TRAIN_SET]
                  [-test_set TEST_SET]
                  [-test_set_predicted TEST_SET_PREDICTED]

optional arguments:
  -h, --help            show this help message and exit
  -smoothing SMOOTHING  yes|no
  -train_set TRAIN_SET  path to train set
  -test_set TEST_SET    path to test set
  -test_set_predicted TEST_SET_PREDICTED
                        path to save the predicted pos sentences
```

Source

https://gitlab.com/Yzoni/natuurlijke-taalmodellen-en-interfaces_2015_2016