Natural Language Models and Interfaces
Part A

Maurits Offerhaus (10400036)
Amir Alnomani (10437797)
Joram Wessels (10631542)

March 1, 2016

Abstract

The goal of this assignment was to implement a markov POS tagger, in two seperate steps. The first step would be a training step in which the trainig corpus is used to estimate the probabilities for the models. Second would be a tagging step in which the Viterbi algorithm is used to calculate the most probable POS tag sequence for a given input sentence by using:

$$argmax_{tags} P(tags|input\ sentence)$$

1 Introduction

The use of language plays a big role in our modern society, the same goes for computers. Because the computer is such an important factor in our lives and used for a lot of different applications, it is necessary to make them perceive our language.
During this course a Python program was written in different steps in order to learn more about language models interfaces and different methods of syntactic analysis.
In this specific assignment part of speech (POS) tagging is used to identify the syntactic categories of all the words in a given sentence. This can be done using several different techniques. First of which would be assigning every uni-gram with their most probable tag, this already gives an accuracy of around 90%
A second technique would be using a bi-gram model to assign the most probable tags. This however will still result in falsely tagged sentences because the model will not look ahead and thus make incorrect choices in the beginning of a sentence which will lead to highly improbable choices later on.
A better technique for tagging a given sequence is using a Hidden Markov Model (HMM). An HMM this will give us the overall most probable sequence of tags of hidden states by using the observed states.
To find the most probable tag sequence T for a given untagged sentence S we use:

$$argmax_T p(T|S) = argmax_T p(S|T)p(T)$$

To find the best path, the Viterbi algorithm is used. This algorithm stores partial results in a chart as to avoid re-computing them.

$$v(j,t) = max_{i-1}^{N} v(i,t-1) \cdot a_{i,j} \cdot b_j(o_t)$$

This algorithm stores a back trace to show which cell it came from to get the most probable tag, when the sentence is fully computed and all the values are stored in the chart, the backtrace will show us the most probable tag sequence.

2 Method

In this final part of step A a POS tagger using the Viterbi algorithm had to be implemented. As advised, the Viterbi algorithm was first implemented by hard coding the values from the lecture slides and the results were compared. If the values outputted by the code and the values on the lecture slides were the same, we would be sure that the implementation of the Viterbi algorithm was correct.

2.1 Ngrams

As mentioned earlier ngrams will play a big role in natural language programming. An ngram is a continuous sequence of length n words from a sentence. The ngrams which are most commonly used are unigrams ($n = 1$), bigrams ($n = 2$) and trigrams ($n = 3$). These ngrams are used to calculate the probability of a sequence of words. This is done by counting the total number of occurrences of the ngram divided by the count of the $n - 1$ gram. For example this is what a bigram model would look like:

$$P(w_i|w_{i-1}) = \frac{P(w_{i-1},w_i)}{P(w_{i-1})}$$

Generally we can say that when we have an ngram model with a larger n, the model will be able to store more context with a well-understood space-time trade off. Also the more training data a model will have, the closer the estimated likelihoods will get to their real likelihood.

2.2 Provided files

For this assignment there were two provided files, which are the corpora. These corpora are text files which consists of tagged words which make up a sentence.Normally a word would have one tag: `Investment/NNP`. In the corpora there were however also words with tags which usually are not used in POS tagging, such as: `7\/8/CD` and `S*/NNPP/NN`. These tags had to also be adapted in the program.
To identify when a word is considered a word, it has to be between two spaces or on the next line. The end of a sequence is given by either:
`==========` or `./.`
It is important that the sentences are separated, because the probability will only span over their own sequence.

2.3 Smoothing

When calculating the probabilities, some problems may arise. There could for instance be ngrams with a frequency of 0, this means that a certain ngram would never be possible because it has no occurrences in the corpus. To fix this Good-Turing smoothing is used to alter the frequency of ngrams by "stealing" a certain value from high-frequency ngrams and giving a certain value to the 0

frequency ngrams. Now even when a ngram does not occur in the corpus, it can still be assigned a probability.
For words that did occur in the test set a new probability has to be calculated because some value of their probability if given to the 0 probability words. These new probabilities are calculated using the Good-Turing method:

$$c^* = \frac{(c+1)N_{c+1}}{N_c}$$

3 Results

The program compiles using Python version 3.5 with the following command:

```
assignmentA.py -train-set [path] -test-set [path]
           -test-set-predicted [path] -smoothing [yes|no]
```

4 Discussion

We can conclude that using the right methods, a reasonable result is achieved for assigning POS tags to words in different sentences. When using smoothing, the time for the program to complete the tagging took considerably longer.
It is obvious that there is still room for improvement, currently even the most state-of-the-art POS taggers "only" have an accuracy of 97-98%. This means that even the best POS taggers still have different levels on which to improve. Some of these improvements would for instance be: a bigger training corpus, or an ngram model with a higher number n. The problem however would be that when using a larger n the amount of calculations will grow exponentially.
In the future the computational power of computers will keep growing and bigger calculations will be possible. This will probably result in POS taggers with an accuracy even higher than 97-98% which means that it will get closer to a fully accurate POS tagger. It is however hard to believe that a POS tagger will reach an accuracy of 100% in the near future because of how ambiguous the human language is.