Practical 1: POS tagging with HMMs

180028981

March 2019

Contents

1 Introduction

For this practical, a first-order HMM (Hidden Markov Model) for POS (Part of Speech) tagging was developed using Python 3.7. Natural Language Toolkit 3.4 was used for the purpose of performing frequency distributions, probability smoothing using Witten-Bell and finally for access to the Brown corpus for training and testing purposes. The two algorithms that were developed are the Viterbi Algorithm and the Beam Search algorithm.

2 Viterbi Algorithm

2.1 Setting Up

The Viterbi Algorithm uses the following formula to calculate the best tag for each word: Where

$$\hat{t}_1 \cdots \hat{t}_n = \underset{t_1 \cdots t_n}{\operatorname{argmax}} \left(\prod_{i=1}^{n} P(t_i \mid t_{i-1}) \cdot P(w_i \mid t_i) \right) \cdot P(\langle /s \rangle \mid t_n)$$

the words from any given sentence are denoted by $w_1..w_n$, and t_0= <s> and t_{n+1} = </s> are the start-of-sentence and end-of-sentence markers respectively. The way the algorithm works is by calculating the Transition and Emission probabilities which are defined as follows:

- **Transition Probability**: `q(s|u, v)` which is defined as the probability of a state "s" appearing right after the observations "u" and "v" in the sequence of observations.
- **Emission probability**: `e(x|s)` which is defined as the probability of making the observation x given that the state is s.

When talking about POS tagging the states are the possible tags that a word can be given and the observations are any given words coming from a sentence that is being analyzed. Before the transition and emission probabilities can be calculated, the frequency distribution for each tag given a word and for each tag given the previous tag needs to be calculated. This is done by counting each instance of these situations and then passing it to the nltk method for calculating a frequency distribution. After that step is complete, the frequency distribution needs to be smoothed to account for changes in the input. This is again done by using the nltk method for Witten-Bell probability smoothing. These methods return dictionaries that we can use in order to check for tags given other tags or words. The algorithm also keeps track of each unique tag used while training and saves them in a separate list, including the start-of-sentence and end-of-sentence markers.

2.2 Implementing the Algorithm

The way the algorithm was implemented is as follows:

1. A 3D List will be used to store all the probabilities and navigate through the best path. This list has the following axis: x - words, y - tags, z - a tuple of the end probability and a pointer which shows which tag from the previous step was used to calculate it.
2. The algorithm checks the first word of a sentence and then calculates the emission probability (e) using the word (s) and each of the unique tags ($x_1..x_n$). The algorithm also calculates the transition probability (q) coming from the state "start-of-sentence". These two values are multiplied and stored into the 3D list with a "start pointer" to signify that there are no words before this point.
3. When the algorithm goes to examine the second word and onward, instead of just multiplying (e) and (q) it also multiplies their result with every possible probability from the previous word examined, and the one that gets saved is the highest out of these values. The pointer now points to the value from the previous step that was used to calculate the value that was just saved.
4. After all the words in that sentence have been examined, the algorithm looks at the last entry and finds the highest value. After that, it follows the pointers backwards until it reaches the starting pointer mentioned in 2. That sequence of tags is the final output of the algorithm

3 Beam Search Algorithm

This algorithm is very similar to the Viterbi algorithm. All the steps are the same except step 2. This algorithm has a parameter called Beam Width (K). K is used to limit the number of tags to be examined from the previous step. For example, if K=1 then the algorithm will only look at the highest value of the tags in the previous step(i.e. Eager Algorithm), if K=2 then the algorithm will look into the 2 highest values and so on. If K is equal to the number of unique tags, then the algorithm is identical to the Viterbi algorithm. The structure of the code remained similar in most cases between the beam algorithm and Viterbi, however the 3D list structure did change to use a dictionary for the middle part of the list i.e. the list is a list of dictionaries where each one given a key returns a list of 2 values: the probability and the pointer.

4 Testing and Results

The accuracy of the 2 algorithms is pretty similar as can be observed below. These are the results with 10000 sentences used for training and 500 for testing as per the specification.

	Accuracy
Viterbi	95.01%
Beam Search K=1	93.32%
Beam Search K=5	94.12%
Beam Search K=Len(UniqueTags)	95.01%

Table 1: Algorithm Accuracy Results

5 Running Instruction

All running instructions are included in the README.md file.

6 References

- Jurafsky, D. and Martin, J.H. (1999), Speech and Language Processing, 2nd Edition, Alan Apt
- Language and Computation Lecture Notes from St. Andrews University, written by Mark-Jan Nederhof
- Natural Language Toolkit Version 3.4