Part-of-Speech Tagging Using Hidden Markov Model

Saber Bahranifard

March 8, 2020

Abstract

The goal of this project is to tag all the words in an unseen Test corpus. To achieve this goal, a tagged Train corpus along with a tagged Development corpus are provided to train and evaluate a model. As model, for this sequential task, a bi-gram Hidden Markov Model is used. The HMM model is optimized using a Viterbi Algorithm. Also, to avoid zero probability for the words in the Test corpus which are not already observed in the training set, a Laplace Smoothing method is used. The Development set is tagged to be able to score the model. The accuracy of the POS tagging is reported at the end of this brief.

1: Code Structure

A Part-of-Speech (POS) tagging is assumed to be a Markov Process with tags being assumed as hidden states and each observed word in a sentence as observed data. The Markov Process is relying on the facts that:

- At each state, the observed data is conditionally independent of previous states and is just conditionally dependent on the current state.
- Each state is just conditionally dependent on the immediate previous state and is conditionally independent of any state beyond or before.

Using Hidden Markov Model (HMM), this Markov Process can be learned in a stochastic approach. The basic components of a HMM model for POS tagging are three distributions:

Initial Distribution, $P_S(T_i)$**:** Probability of a sentence in the corpus begins with T_i in the Tag Set.

Transition distribution, $P_T(T_j|T_i)$**:** Probability of a transition to tag T_j given the current state is tag T_i

Emission distribution, $P_E(W_j|T_i)$**:** Probability of observing the word W_j given the current state is tag T_i

The purpose of the POS tagger is to find the most likely sequence of tags for a given sentence. For this purpose, we need to maximize the joint posterior distribution of latent variables, in this case tags, given a sequence of observations, in this case words in a training corpus. To be able to implement such optimizer on the HMM model, a dynamic programming algorithm, called **Viterbi Algorithm** is used. This algorithm takes in the aforementioned distributions and iteratively finds a sequence of tags which maximizes this probability.

In the provided code, inside the `HMM_tagger.py` file, a HMM class is defined which does all the word counts required to setup the Initial distribution `log_init_dist`, Transition probability matrix `log_tran_dist`, and Emission distributions `log_emit_dist`. Also, to avoid overfitting to the training set and under-weighting the unseen words, a Laplace Smoothing method is used. The (`Laplace_smoothing`) method under the HMM class

is responsible to form the distributions out of word/tag counts in addition to smoothing these distributions. `HMM_numpy`, inherits the distributions from the `HMM` class, and using `Viterbi(sentence)` method, calculates optimal sequence of tags for a given sentence. This method returns the most probable sequence.

At the top of the code, the `HMM_main.py` has to be called by `python` to call `HMM_numpy` object. This file, in addition to calling the `HMM_numpy` object which respectively setups HMM distributions and Viterbi algorithm, it reads and writes sentences from and to file objects. The syntax to call this code, wether you want to evaluate the model on *dev set* or test it on *test set* is as below:

Evaluate the model:

```
$ python ./main_HMM.py training_file dev_file
```

Test the model:

```
$ python ./main_HMM.py training_file dev_file test_file
```

To run the code, make sure *python 3* is used.

2: Results

Using a training set of size 39,832 sentences the HMM model is trained and then evaluated on a development set of size 1,346 sentences. The code generates POS tags for the dev set and then the provided `scorer.py` code is used to score the model. The scorer reports an accuracy of **91.9%** for the model being evaluated on the *dev set*. For the un-tagged test set, a `POS_test.pos` file is generated which has all the inferred tags for the words in the *test set*.