ELL881 ASSIGNMENT-1 REPORT

By: Ritika Soni (2020MT10838)

Hidden Markov Model

To create a vocabulary for HMM, I preprocessed the tagged training data. Words from the training set that were used twice or more are included in the vocabulary. Because the POS tagger would inevitably come across words not in its datasets, I added a few more terms to the vocabulary (set of "unknown word tokens").

These words are also examined during preprocessing to extract any indications that may be useful in tagging them appropriately, hence increasing accuracy. The suffix "ize," for instance, indicates that the word is a verb in phrases like "final-ize" and "character-ize."

In the training and test corpus, a collection of unknown tokens, such as '--unk-verb--' or '--unk-noun--', will take the place of the unknown words and appear in the emission, transmission, and tag data structures

For constructing the Transition and Emission matrix, I have used smoothening as discussed in class.

An entry of the transition probabilities matrix A is given by:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i) + \alpha}{C(t_{i-1}) + \alpha * N}$$

An element of the emission probabilities matrix B is given by:

$$P(w_i|t_i) = \frac{C(t_i, word_i) + \alpha}{C(t_i) + \alpha * N}$$

The naive Bayes algorithm uses smoothening to determine the conditional probability of an event given a class label. On the other hand, the conditional probability in testing data will be 0 for the duration if a new event occurs.

In this assignment, I used the prior word to guide us through a corpus to anticipate POS tags. Bidirectional POS tagging is used in various ways. To anticipate the POS tag for the current word in a corpus, bidirectional POS tagging necessitates knowledge of both the preceding and subsequent words. Instead of merely knowing the previous word, bidirectional POS tagging would provide additional information about the POS.

Maximum Entropy Markov Model

I used the various features for MEMM, but the below features were my go-to when attempting to raise the feature-based tagger scores.

My model has the following characteristics: (1) the current word xi; (2) the two predictions that came before it, t i-1 and t i-2; (3) prefixes and suffixes of xi; (4) the following two words, xi+1 and xi+2; (5) the current word's POS tag; and (6) person lexicons.

The MEMM tagger consists of training and prediction components, the same as the HMM tagger.

To train a model, I first extract training data in a manner that a log-linear trainer can comprehend from the training corpus, which is a three-stage process.

Stages are:
1. Feature extraction: After reading the train corpus, create a feature vector file with the following line appearances for each line:

label [feature1] [feature2]...[featureN],

where everything is made of stings. For instance, if the word form, the three-letter suffix, and the preceding tag are the features, we could have lines like

NN form=walking suff=ing pt=DT,

showing a situation where the word is walking, its suffix is ing, the gold label is NN, and the previous tag was DT.
This approach was mentioned in the paper given below:
https://www.researchgate.net/publication/2476010_A_Maximum_Entropy_Model_for_Part-Of-Speech_Tagging#:~:text=A%20maximum%20entropy%20based%20tagger,the%20paper%20Ratnaparkhi%20Table%202.

2. Feature Conversion: Utilising the features file generated, create a file in a format readable by the log-linear trainer.

3. Model Training: create a trained model file by introducing a model using the training file that was previously generated. We may use any solver we like here, but I used a Multinomial logistic regression solver.

Next, I employed the model I had trained for the preceding task within the Viterbi decoder.
Though I have written the code, the model is taking a long time to train. Additionally, there were hyperparameters in logistic regression that needed to be adjusted. My ignorant method could be another factor. Sentence-by-sentence analysis of the data and word-by-word tag prediction are the apparent methods of predicting a tag for every word in the dataset while referencing the previous tag predictions.

This resulted in more calls to the sklearn logistic regression classifier's prediction function, which took longer.

I also noticed that it helps to consider two distinct information sources while retrieving POS tags. First, words exhibit statistical predilections according to their part of speech. Second, a word's part of speech is greatly influenced by its context.