Hidden Markov Model

The below program aims to implement a Hidden Markov model using the Viterbi algorithm. There are certain morphological features that have been implemented in the HMM in order to improve the accuracy of assigning an accurate POS tag to unknown words.
The final accuracy reached for the development corpus through the score function provided to us was
31490 out of 32853 tags correct
accuracy: 95.851216

In order to reach this accuracy I have handled morphology through 2 techniques. This write up will also list down the accuracy of individual technique and the final combination which led to the above mentioned accuracy

Morphology using regex pattern Matching (morph function in morphology.py)

In this technique I handled unknown words through pattern matching. I used regex library in python to implement pattern matching. These patterns were observed in the training corpus as well as general POS rules. Below are the regexes for handling certain POS
[A-Z] – NNP
'-' - JJ
*ion , *ty, *ics , *ment , *ence ,*ance ,*ness ,*ist ,*ism – NN
*ate, *fy, *ize, *en, *em, *ing – VB
*un,*in,*le,*ry,*ish,*ous,*ical,*non – JJ
(Rest of the regexes can be found in the morph() function in morphology.py)

The words which could not be identified were tagged as 'UNK'

Morphology using prefix and suffix matching (morph_advanced function in morphology.py)

In this probabilistic technique, the prefix and suffix of the unknown word was extracted and then was matched with existing prefixes and suffixes in the training corpora to determine the POS tag of the unknown word. The algorithm is as follows

Training

1. Extract prefix of all the words in the training corpora till length prefix_optimal. Store the tags associated with this prefix along with the frequency with which the prefix occurred in that tag
2. Calculate the probability associated with each tag which is mapped with the prefix.

3. Find out the maximum likely estimated tag which can be associated with a particular prefix by finding out the tag with the maximum probability and store it in a map with the prefix as the key and the tag and the probability as values.
4. Repeat the same steps for the suffixes for all training words.

Algorithm

1. From length minimum_prefix_length till prefix_optimal_length extract the prefix of the unknown word for every length
2. Find out the tag and the probability associated with the prefix from the map created in the training step and store it in a list.
3. From the list created, find out the tag which occurred with the highest probability. Store it as prefix_tag,prefix_probability
4. Repeat the same steps with the suffix of the word
5. Compare prefix_probability and suffix_probability, if prefix_probability was higher return prefix_tag else return suffix_tag

After experimentation as tuning the following values gave the highest accuracy score

minimum_prefix_length = 2

minimum_suffix_length = 2

prefix_optimal_length = 6

suffix_optimal_length = 15

If the tag was not found through this method ‘UNK’ was returned

Combining Approach 1 and Approach 2

The highest accuracy was observed when the approach 1 and approach 2 were combined. For simplicity the first approach will be referred as morph and second approach would be referred as morph_advanced.

There were two ways of combining the two approached

1. The unknown words were passed through morph and the words which were tagged as ‘UNK’ were then passed through morph_advanced. If the tag still came out to be ‘UNK’, a minimum emission probability was assigned to the word along with a minimum transmission probability.
2. The unknown words were passed through morph_advanced and the words which were tagged as ‘UNK’ were then passed through morph. If the tag still came out to be ‘UNK’, a minimum emission probability was assigned to the word along with a minimum transmission probability.

HMM

There were basically 5 variants of HMM which were tested. In order to avoid overflow situations, logarithm of the probabilities were used throughout. In order to remove the errors created by zero probabilities, a minimum transmission probability and a minimum emission probability were assigned to each word. These were calculated using experimentation and tuning.

The five algorithms are

- Baseline algorithm – Unknown words not handled and assigned a minimum probability
- morph – Unknown words handled through morph approach only
- morph_advanced – Unknown words handled through morph_advanced function only
- morph => morph_advanced – Unknown words handled by first filtering the words through morph approach and then tagging the unknown words through morph_advanced approach.
- morph_advanced => morph - Unknown words handled by first filtering the words through morph_advanced approach and then tagging the unknown words through morph approach.

Algorithm	Development Set accuracy
Baseline	92.7%
Morph	95.629%
Morph_advanced	95.571%
Morph=>Morph_advanced	95.851%
Morph_advanced=>Morph	95.574%

Thus Morph=>Morph_advanced algorithm gave the best accuracy on the development set.

Folder

The zip folder consists of 6 files

main.py – Code for training, validation testing and testing.

Viterbi.py – Code for the viterbi algorithms

utilities.py – Utility functions used

preprocessing.py – Code for creating dictionaries, transition probability map, emission probability map, prefix and suffix probabiliity map.

Morphology.py – Code for implementing morphological approach as discussed above

score.py – Code for calculating accuracy