Assignment 2
Natural Language Processing

Name : Shravan Shetty
Email : shravan.shetty@colorado.edu
Fall 2018

Data Representation

- Scanning through the file I have created a dictionary inside a dictionary to store each words and corresponding counts of tags associated with that in the training data.
  - E.g. {“Shravan” : {“NN”:1 , “VB”: 2 ...}}
- Created a dictionary to store the count of tag given a previous tag. Key for the dictionary in this method is created from the combination for 2 tags separated by “|”. While fetching this count I take 2 tags as arguments and then create a key and search for the count. Zero is returned if the key is not found.
  - E.g. {“VB|NN” :2}
- Created unique list for both words and tags.
- Created dictionaries to keep track of the counts of words and tags.
- Used numpy for creating matrices.
- Used “<s>” as starting tag and “<unk>” to represent unknown words.

Training/Test Distribution

Taken the complete training data given and used the first **80%** of the sentences for training and rest **20%** for testing

Baseline Method

As a part of baseline implementation, While training, I have taken the counts of all the words and respective tags given by the training data. Later while testing took each word separately and assigned the most common tag to that according to the counts from training set. If in case word is not present then that is taken as an error. I achieved around **86%** accuracy with this method.

Handling Unknown Words

Tried to handle the unknown words in 2 ways.

1. Added "<unk>" to the end of the list and considered it as any other word. As I have used add one (Laplace smoothing), the probability of word given any tag will not be zero. Likewise in the test set, any word not in my unique word list from the training is replaced by "<unk>" and then passed to Viterbi. This will give an accuracy of about **87%** .
2. Replaced all the words with a frequency lesser than n by "<unk>". The value of n is formulated from looking into different word frequencies. Looking into the distribution I have taken the value of n as 10. Decided on that number after varying it few times between 1 to 30. I achieved an accuracy of about **90%** using this. (After smoothing in Viterbi)

I have taken the second method to implement Viterbi.

Smoothing

While creating the bigram model for word given tag and tag given previous tag, I observed that most of them have value zero. Now to handle this I have used **add one smoothing** (Laplace smoothing) by adding one to all the bigram counts before the probability is calculated. In the code this is being handled as a separate parameter as smoothing factor and we can test by changing that as well.

Implementing Viterbi

Implemented Viterbi after doing Laplace smoothing and handling the unknown words. In addition to that to run through Viterbi, I created 2 matrices to keep track of probability of a tag given a previous tag (tagvstagMatrix: transitional probability) and word given a tag(tagvswordMatrix). To create these matrices I have used my unique word list and unique tag list as rows and columns. For E.g. if VB has the index 3 in the unique tag list tagvstagMatrix[3,3] will be the probability of VB given VB as previous tag.

After doing all the prerequisites for the algorithm, Following all the few conventions used in the Viterbi algorithm (Viterbi Method in the code).

- Viterbi method accepts 5 parameters rows, columns, viterbiMatrix and wordList. It returns the list of indices of the tags in the unique tag list as output for a given sentence.
- For each test sentence, I go through each words and replace it by "<unk>" if it is not present in the unique word list from the training. Now this new list of words for a given sentence is sent as wordList. The length of this list is sent as columns.
- The length of the unique tags without starting tag is sent as the row.
- Both of viterbiMatrix and backtrack matrix will have the same dimension.

- First column of both the matrices are filled in the first part of the method . Value of the each cell is the product of the probability of the first word given each tag (observation likelihood) and transitional probability of the each tag given <s> as the previous tag. The first column of backtrack matrix will be assigned to zero.
- Rest of the columns are calculated by taking the product of the following values
  - Previous Viterbi path probability.
  - Transitional probability a tag given a tag.
  - Observation likelihood of word given a tag.
- While filling each cell of the viterbiMatrix, recorded the index of the previous row of the column in the backtrackmatrix.
- Once the matrix is populated the maximum of the last column is taken and best path for the same is derived from the backtrack matrix. Now this will be returned as the list of indices for the tags.
- This gives an accuracy of about 90%.

Conclusion

Using Viterbi I get an about **4%** better accuracy on my test without randomly separating the train and test data. I also observed that after randomly taking the training set and test I got about **95%** accuracy. However this should not matter when we test against real test set.