BBM497: Introduction to Natural Language Processing Lab. **(Due: 19/04/2020)**

Submission Assignment #2

Instructor: Necva BÖLÜCÜ *Name:* Doğukan Berat KARATAŞ, *Netid:* 21527142

1 Introduction

The purpose of this assignment was to create a model of hmm with the "train" data we have and try to guess the tags of the sentences in our "test" data set using this model.

2 Task 1: Build a Bi-gram Hidden Markov Model (HMM)

In this section, I had to create 2 probability sequences to create an HMM model. These were the transition probability of all tags created according to the bi-gram and the emission probability created according to the tags to which all words are linked. Actually, there is a need for an initial probability that a tag is kept at first, but I did not create an initial probability sequence since I added the '¡s¿' (start) token to the beginning of the bi-gram transition tags when I was building the structure.

2.1 Transition Probability

I create 4 structures:

- **each_tag_counts**; I calculate the count of each tag in the train data set and add the result to this array.
  Example: {'B-ORG':185, 'O':35007, 'B-MISC':87, ...}
- **transition_tags**; I reprocess each tag in a sentence according to the bi-gram model and add the result to this array.
  Example: [['$< s >$ B-ORG', 'B-ORG O', 'O B-MISC'], ...]
- **transition_tag_counts**; I calculate the counts of each pair in transition_tags and add the result to this dictionary.
  Example: {'$< s >$ B-ORG':1250, 'B-ORG O':245, 'O B-MISC':56, ...}
- **transition_prob**; Finally, I calculate the probabilities of each pair in transition_tags and add the result to this dictionary.
  Example: {'$< s >$ B-ORG':0.55, 'B-ORG O':0.04, 'O B-MISC':0.0023, ...}

2.2 Emission Probability

I create 3 structures:

- **emissionTag_Word_dict**; I classify each word in a sentence according to the tag to which it is linked and add the result to this dictionary.
  Example: {'B-ORG': ['EPR', 'EU', 'European', ...], 'B-LOC': ['BRUSSELS', 'Germany', ...], ...}
- **emission_word_counts**; The value part of each tag in emissionTag_Word_dict is the word list with that tag. Here I calculate the counts of the words in this list and add the result to this dictionary.
  Example: {'B-ORG': {'EPR':3, 'EU':5, ...}, 'B-LOC': {'BRUSSELS':12, 'Germany':3, ...}, ...}
- **emission_prob**; I calculate the probability of the words in emission_word_counts structure and add the result to this dictionary.
  Example: {'B-ORG': {'EPR':0.2, 'EU':0.4, ...}, 'B-LOC': {'BRUSSELS':0.4, 'Germany':0.1, ...}, ...}

3 Task 2: Viterbi Algorithm

I tried to guess the tags of the sentences in the 'test' data set using the HMM model we created in this section. I used the matrix structure to make this prediction easier.

3.1 Keeping The Rows Of All Matrices In The Same Order:

Now I have 3 matrices. I create Transition Matrix only 1 time. I create Emission and Viterbi Matrices for each sentence. That's why the tags that make up the lines of the Transition Matrix and the Emission and Viterbi Matrix don't come in the same order. I sort the tags to solve this. In this way, I get tags in the same order for each sentence.

3.2 Changing Probability Dictionaries to Probability Matrix:

To calculate the Viterbi algorithm, I have converted dictionaries with transition and emission probes into a matrix.

3.3 Calculate Viterbi

First of all, I calculate emission prob for each sentence from test_sentences and add them in the emission_matrix, as I mentioned above. Next, I fill each cell in the viterbi_matrix with the formulas in the below: (The i, j, k values are specified in the comment section of the viterbi() function.) – [line #305]

- In the first step, I multiply the probability that any tag from transition_matrix arrives initially and the probability that the first word is inside that tag, and I add the result in the first column of the Viterbi matrix, respectively.
  Formula: transition_matrix[j][i] * emission_matrix[j][i]
- In other steps, I apply the following formula:
  Formula: viterbi_matrix[k][i-1] * transition_matrix[j][k+1] * emission_matrix[j][i]
  - viterbi_matrix[k][i-1]: Probability of come the previous tag in Viterbi matrix.
  - transition_matrix[j][k+1]: The probability that the tag in the cell that it is in comes after the tag I chose in viterbi_matrix[k][i-1].
  - emission_matrix[j][i]: The probability that the word in the step it is in is inside the tag in the cell where it is located.
- In this way, I fill each cell with the total number of tags, take the highest of them and write them to viterbi_matrix[j][i].
- Then I add the tag corresponding to the highest value in each column of viterbi_matrix, which I filled with these calculations, to the predict_tags list.
- Now I have guessed tags. Finally, I send this to the accuracy() function and measure the accuracy of the model.

4 Task 3: Evaluation

Finally, I compare the tag list of each sentence I guess with the Viterbi algorithm with the actual tags of that sentence I get from the 'test' data set. And in this way, I calculate the accuracy rate of my model.

4.1 Which Process Has Increased Accuracy And How Much?

- At first, I encoded and ran all functions without any operation. Accuracy = %14
- Then I noticed that the words in the test sentences were not in the train data set. Considering this situation, I performed smoothing. Accuracy = %51
- Then, as I mentioned in 3.1, I realized that the tag order in the lines of emission and Viterbi matrix may be different in the tag order in the transition matrix. And while creating these structures for each sentence, I gave the tags in order. Accuracy = %86

- Finally, when I was reading a file, I was converting all words to lowercase. I tried without converting to lowercase. And the final Accuracy output = %90.23.

As a result of these transactions, my current correct estimate rate %90.23.