BBM497: Introduction to Natural Language Processing Lab. **(Due: 20/04/2020)**

Submission Assignment #2

Instructor: Asst. Prof. Dr. Burcu CAN, Necva BÖLÜCÜ *Name:* Deniz Zağlı, *Netid:* 21527668

- Implementation and Detail:
  - **preprocess(lines):** In the preprocess function, necessary operations are performed before using dataset. One of these processes is to delete the spaces in the dataset. Another process is to separate the articles given the starting parameters. Make lowercase letters so that the initial letters of the words do not differ. All these operations take place in the preprocess function. Due to the convenience of the functions, I used the list data structure while performing these operations.
  - **find_count_of_unique_words(sentences):** I created this function for smoothing. This function finds the total number of different words in the given dataset.I used a set to find out how many. The set helped me extract the same words.
  - **HMM_Model(sentences, count_of_unique_words):** This function creates the hidden markov model requested from us. As a result, it returns the dictionaries containing Initial, transition, emission values. I will explain how these dictionaries were created in order. While calculating the initial values, the entire dataset was visited and the tage in each line was checked. If this tag is not in the dictionary, it is added to the dictionary and its value is given as 1. If there is in the dictionary, its value is increased by 1. In this way, the frequency of the tags was calculated. Then probability values were calculated for the tag values whose frequencies are known. For ease of calculation, the logarithms of the probability values in the base of the calculated values are taken. For the transition process, the dataset was visited as a bigram. Binary word groups that are created, if not, are created and equal to 1. If it is in the dictionary, its value is increased by 1. I combined these binary phrases with string operation for convenience in search. I created a key as a result of this merge. The probability distributions of the values with frequencies, and then logarithm values on the base 2 were found. For the emission values, all datasets were visited in order. Key is created by combining the string operation with the string operation for the convenience of searching the word and the tag that the word is linked to. If this key is not in the dictionary, it is created and assigned a value of 1. If there is in the dictionary, its value is increased by 1. After the frequency values found, probability distributions were calculated and logarithm of the values found on the base of 2 was taken. Laplace smoothing was applied to solve the zero probability problem while creating the values in the whole dictionary. While performing these operations, I used a dictionary for ease of access and a list for ease of application.
  - **viterbi_algorithm(sentences, initial, transition, emission, initial_smoothing, transition_smoothing, emission_smoothing):** The viterbi algorithm I created has 2 different working states. The first case and other cases. In the first case, we need an initial value. This value is the initial value we get from the hidden markov model. I found and multiplied the probabilities of these tags and the probabilities of words depending on the tags. The index of the highest one among these values I gave gave us the tag of the first word. In the other part, by accepting what the previous tagin is, we calculated the probability of probability and multiplied this probability value by the probability of the words linked to the tags. As a logarithm, we actually collected it because we performed these operations. The index of the highest value gave us the tag of our new word. I did this for all writing groups. In cases where the word has no equivalent in any probability table, I applied laplace smoothing. While performing these operations, I benefited from the list data structure because it provided me ease of operation. I used the math library for logarithm operations.
  - **dataset(folderpath):** Opens the file in the path it takes as a parameter. Assigns the lines of the file to a list.
  - **accuracy(gold_sequence, predicted_sequence):**It shows the result of how many percent of these estimates made by making predictions on the test dataset is correct. The accuracy percentage of my algorithm is 85%.