




Attachments:

 distilBERT.ipynb

 multilingualBERT.ipynb

 parsBERT.ipynb

Preprocessing: I open the Train.txt file. I count the number of tags and separate the words from sentence to sentence. From these sentences, I make a two-column dataframe, the first column contains the tokenized sentence and the second column contains the tags or tags of these words in this sentence. 23 tags counted. Like each of these, I put a number. This work is done with the `preprocessing.LabelEncoder()` model from sklearn and can be returned with the same model. All these operations are done on the test data and its first column is entered into the network for pos tagging after setting the model. I divide the data frame I made from the training data into two parts: training and validation.

Because the network opening has a fixed size and some sentences may have less or more words than that, I tokenize the sentences with a function that converts the output size, which is actually the size of the network opening, to the number of words. and map their labels. `tokenize_and_align_labels` is used as a tokenizer in this network function that is going to be fine-tuned.

In this exercise, I used three networks. I did the following steps for each of these in separate files.

1. I loaded the grid with `AutoModelForTokenClassification`. This class sets the network so that its output layer gives an output vector equal to the number of classes (number of tags) for each input token.
2. I adjust the fine tuning parameters. For `learning_rate` I tried numbers between 0.0001 and 0.0000001. First, I made the networks with `num_train_epochs=3`, but by increasing this number, the results got better and better. And 10 to 15 gave the best results, and after that there was not much improvement on the validation loss. Because the last feedforward layer of these networks is supposed to perform the classification of tokens, the main part of fine tuning works on this layer, and that is why, unlike exercise 5, more IPACs are needed to get a good result. In the previous exercise, the main focus was on a linear classifier or a feedforward with one output (lower weights).

3. I do the tagging with the fine-tuned network. The result of the work for each sentence of the test data (the pre-processing mentioned above was done on Test.txt) is a vector with the size of 23 for each input token. argmax is taken on each of these and the labels of the tokens are determined.
4. Calculation of accuracy on test data

The networks I used for training:

- distilbert-base-uncased: a language model trained with data from Wikipedia and English books and magazines. And a fine-tuned model of it is also available, which is fine-tuned for NER and has given good results. (Its language model is used here.)
- HooshvareLab/bert-fa-zwnj-base: It is the same model as Persian bert or parsbert.
- bert-base-multilingual-cased: a language model trained with data from 104 different languages.

Because I am going to make comparisons with the data of exercise 3 and in this exercise only accuracy was calculated, I will limit myself to the same criterion here.

Viterbi – HW03	82
distilBERT	76.5
multilingualBERT	88.7
parsBERT	81.8

The accuracy of Persian bert is equal to Viterbi algorithm. The model that was trained only with English data has a weaker performance for Persian language, but the multilingual model that was trained with 104 different languages has better accuracy even than Persian bert and Viterbi. Compared to Viterbi, definitely because the model can learn more complexities of the language. And the reason for the higher accuracy compared to Persian bert can also be interpreted that because more languages have been seen and learned, this information can be useful for labeling Persian sentences. In addition, the amount of data with which it has been trained is more than Persian bert and it has a stronger language model for tagging sentences.