NLU P1: PoS Tagger

Ovidio Manteiga Moar

Carlos Villar Martínez

November 2022

1 Introduction

In this assignment we have to develop a neural PoS sequence labeling model that labels the words of a given sentence according to their morphological information. This PoS Tagger is based on implementing layers, the most important one is a neural model based on long short-term memory networks (LSTMs).

The main functionalities that our program must support are: training a model, evaluate a training model and implement a function to compute the part-of-speech tags for given new, previously unseen sentences inputted by the user

2 User manual

2.1 Set up

The Python version used is Python 3.9.13.

To set up the required dependencies, run the following command:
    pip install -r requirements.txt

2.2 Run the program

To run the program use the following command:
    python main.py [arguments]

See usage section below for more information on the available command line parameters.

2.3 Command line interface usage

Run the program with the -h option to see the help:
    python main.py -h

Usage: main.py [-h] [-train] [-validate] [-eval] [-predict] [-verbose] [-model MODEL] [-tag] [-dataset_file DATASET_FILE] [-sentence SENTENCE]
[-dataset UD_English_EWT, UD_Spanish_AnCora, UD_French_GSD, UD_Galician_CTG]

Optional arguments:

1-h, -help: Show this help message and exit.

2-train: Train the selected model.

3-validate: Validate the selected model with the validation set.

4-eval: Evaluate the selected model with the test set.

5-predict: Use the model to predict tags from one of the sets.

6-verbose: Verbose mode to output extra execution info.

7-model MODEL: The model to use: either base or char to use the char embedding.

8-tag: Predict the POS tags for a given sentence. To use with the -sentence parameter.

9-dataset_file DATASET_FILE: The path to the local CoNLL-U dataset files without the -dev.conllu extension.

10-sentence SENTENCE: The sentence to be POS-tagged when "-tag" is passed.
11-dataset: The preconfigured dataset to use, which is automatically downloaded. One of:
UD_English_EWT
UD_Spanish_AnCora
UD_French_GSD
UD_Galician_CTG

2.4 Run in Google Colab

Create a new notebook in Google Colab or import the provided one NLU_P1.ipynb.

Connect to the runtime.

Copy all the Python files (*.py) to the runtime base folder. Run the notebook.

3 Implementation decisions

As far as architecture is concerned we don't really have a lot of freedom as it is fully described on the statement of the assignment, but we do have some freedom on our implementation decisions.

One of the most important things we decided was to create two separate models which are the base model (just using word embedding) and the character embedding model (using word embedding and character embedding), this allows us to run the program with the one we prefer and also to compare the results obtained with both of them. If we look at the structure of both of them we see they are similar, it's important to notice that we avoided using the keras.preprocessing.text.Tokenizer as it doesn't work with tensors and it cannot be added as a layer of the model, so we used the TextVectorization for the word embedding and for the character embedding we defined a function called words to char array which basically extracts the character of our words and their length. We also define a function called ignore class accuracy that makes our model ignore the BLANKS when measuring the accuracy as they can create some 'noise', but this last one isn't really necessary because it yields the same results as using the Mask_Zero = True.

Another decision was that we encoded the targets as integers and used sparse_categorical_crossentropy so we avoided using the OneHot encoding. Then we created a decoder to get the classified tags from the outputs.

A class called POSTaggerConfig was created where we can easily change some of the parameters of our program, like the number of epochs or the maximum length of the words (we decided max word length=20 as there is no indication for this value in the statement of the assignment) for the character embedding. Here we also declared that the configuration in Google Colab and in local are different because if we use in local the same configuration than in Colab it takes a crazy amount of time to run the program.

Another notable thing is the creation of a separate file to download and manipulate the datasets. It's important the class DatasetParser where we parse, tag and pad the datasets, we need to import parse from conllu in order to be able to do this class. There is also included the option of uploading our own file (explained in the manual).

4 Results and analysis

The main idea of this section is to gather the results obtained from the different models and the different datasets. With this information we can also get to some conclusions. To do this we have to be sure about what parameters are important to take into account, some of them are the accuracy, the validation accuracy an the test accuracy.

We have used four different datasets (UD English EWT, UD Spanish AnCora, UD French GSD and UD Galician CTG) and two different models (without character embedding and with character embedding), as expected for diffeternt datasets and for different models we obtain different results.

Before talking about the results obtained we need to know the size of our datasets because it has consecuences on our results. The shape of the UD English EWT is [12542, 128], the one for the UD Spanish AnCora is [14282, 128], in the case of the UD French GSD we have [14446, 128] and for the

last dataset which is UD Galician CTG the shape is [2272, 128]. It's easy to realize that the Galician dataset is much smaller than the other three, in the paragraphs below we'll see what consecuences this has.

For the English dataset with the first model the accuracy was 0,9920 while the validation accuracy was 0,8752 and the test accuracy was 0,8727. In the case of the second model we got an accuracy of 0,9998, a validation accuracy of 0,9049 and a test accuracy of 0,9086. If we look at the numbers we can clearly see that for this dataset the implementation of the character embedding helped to raise our validation and test accuracy. It's also important to take into account the time that each model takes to obtain the results, while the first one takes approximately one minute or minute and a half (using Google Colab) the second takes easily five or six minutes. It's up to us deciding if this time is worth it or not.

In the case of the French dataset with the first model the accuracy was 0,9980 while the validation accuracy was 0,9290 and the test accuracy was 0,9363. In the case of the second model we got an accuracy of 0,9999, a validation accuracy of 0,9276 and a test accuracy of 0,9288. If we look at the numbers we can see that the implementation of the character embedding didn't raise our validation and test accuracy.

In the case of the Spanish dataset with the first model the accuracy was 0,9984 while the validation accuracy was 0,9261 and the test accuracy was 0,9265. In the case of the second model we got an accuracy of 1, a validation accuracy of 0,9247 and a test accuracy of 0,9257. If we look at the numbers we can see that the implementation of the character embedding didn't raise our validation and test accuracy.

In the case of the Galician dataset with the first model the accuracy was 0,9940 while the validation accuracy was 0,4349 and the test accuracy was 0,9266. In the case of the second model we got an accuracy of 0,9984, a validation accuracy of 0,4317 and a test accuracy of 0,9425. One thing that draws the attention is the low validation accuracy this dataset drops in comparison with the other three datasets, there are some factors that can be causing this, maybe our model doesn't fits for this idiom, but probably it's just because of the size of the dataset, being smaller than the other three makes that our program can't be trained properly causing a poor validation accuracy. The size also makes that the times for this data are much lower than for the other ones.

Apart from this two models we also tried some things off script for the Engish and the Galician datasets. For example a remarkable positive change was adding a bidirectional layer to our base model as it made that the validation accuracy went from 0,8752 to 0,9042, this change had almost no effect on the Galician data. Another thing with impact was applying one more dense layer but this made that the accuracy, validation accuracy and test accuracy dropped by a half of their value with just one dense layer. We also tried some things that didn't have a real impact on the results, changing the max word length to 10 and to 30, and also modified the embedding size from 128 to 256 but we got almost the same results that we had before this changes, one important thing to mention is that by lowering the max word length from 20 to 10 it took a bit less time to obtain the results while they were barely modified.

Looking at the results the main conclusion we can reach is that the model with the character embedding is not worth it, this is due to the huge amount of time it takes to run and the results are pretty close to the ones obtained by the other model, in fact, if we use the base model with the bidirectional layer the results are almost the same.