Practical Assignment 1 - Part-of-speech (PoS) Tagging

Sara Micol Ferraina, Adrian Orozco Rivera, Francisco Laport Lopez

General Description

The project has been developed using Google Colab and it is composed of three main files:

1. **Utils.py**: This file contains all the functions needed to download the datasets from the GitHub site and to prepare the data for its classification with the network. Each function has its own documentation, so you can find a brief description of its functionality in the file. Some of the functions you can find in this file are: download_file(args), parse_conllu(args), upos_vectorizer(args), build_word_vectorizer (args) and prepare_data(args).

2. **Tokenino.py:** This file contains two important classes for the project:
    a. **class Token**: represents a token in the CoNLL-U format. This class stores various attributes of a token, such as its part-of-speech tag, lemma, and syntactic dependencies, and provides methods to convert the token into a CoNLL-U formatted string. When parsing the data sets in function parse_conllu(args), one Token object is created for each token of the dataset.
    b. **class MyTagger:** class is a neural network-based part-of-speech (PoS) tagger. It constructs, trains, evaluates, and visualizes the performance of a bidirectional LSTM model designed to predict PoS tags for sequences of words.
3. **main.pynb:** Python notebook with the main code. As has been already mentioned, the project has been developed using Google Colab. The code is organized in three main steps clearly documented in the notebook.
    a. **STEP 1:** Download and parse the datasets.
    b. **STEP 2:** Prepare the data for the network.
    c. **STEP 3:** Build the network, train it and evaluate it.
    d. **STEP 4:** Do the predictions.

How to run the code

Taking into consideration the previous files and that we are working on Google Colab, we must perform the following steps to run the code:

1. **Create our working folder.** Create a new folder in Google Drive and copy the utils.py and Tokenino.py files. E.g., '/content/drive/MyDrive/Projects'

2. **Mount Google Drive.** For this purpose, we must run the following code:

```
from google.colab import drive
drive.mount('/content/drive')
```

3. **Define the working folder path in the code.** We need to declare a variable named working_folder with the path that of the folder created in Step 1. E.g.,

```
working_folder = '/content/drive/MyDrive/Projects'
```

After these 3 steps, the rest of the notebook's code can be run normally without any modifications in a sequential manner.

Implementation Choices

In this section, the main results for the implementation are listed:

1)Standardize=None

We decided to set the parameter standardize=None in the text_vectorizer layer. This disables any pre-processing of the text, such as removing punctuation or converting letters to lower case. By default, TextVectorization performs a basic standardization function that removes punctuation marks and converts all letters to lower case. We performed this because by removing this pre-processing, we noticed that the predicted results were better.

2)Dropout Layer

In addition, we decided to add a Drop out layer (with a percentage of 0.2) after the Embedding layer, this to try to reduce overfitting and improve the generalization capability of the model.

3)Number of epochs

Others variations that were explored on the models were the amount of epochs for the training process and the batch size. The main consequences of these variations are in the amount of time it takes to process the training.

Also, by increasing the amount of epochs it can be possible to cause overfitting to the model because it is shown the same data multiple times causing it to memorize and lose it’s generalization ability.

The final values chosen are 10 epochs and a batch size of 128 because the application runs fast and doesn’t overfit.

Results

In this section, the main results for the implementation are listed.

Training, Validation and Testing Results for English dataset

Table 1 summarizes the accuracy for Training, Validation and Testing datasets used for the network.

Table 1: Training Results for English

Training	Validation	Test
0.9917	0.9893	0.9896

Training, Validation and Testing Results for Galician dataset

Table 2 summarizes the main accuracy values for the Galician Dataset

Table 2: Training Results for Galician

Training	Validation	Test
0.8096	0.8043	0.8081

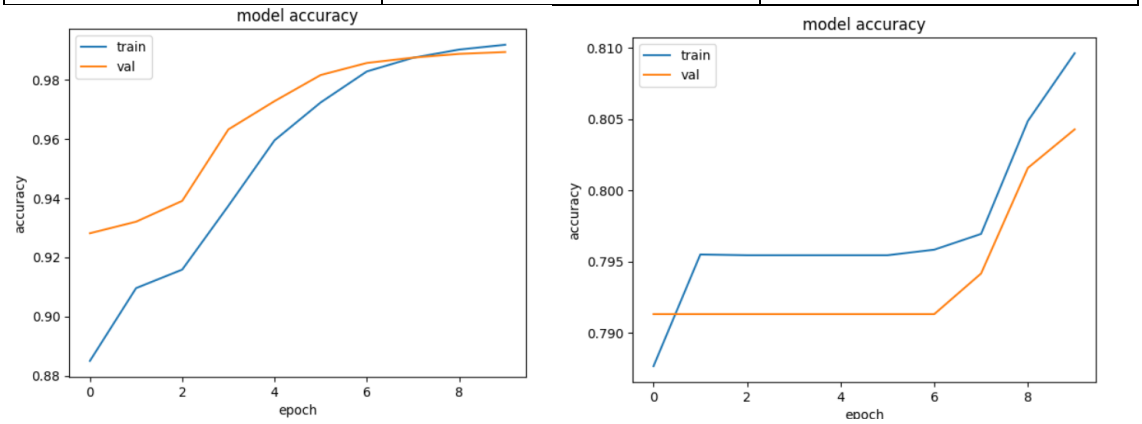

Figure 1: Galician vs English Training Results Plot

Results Analysis

The training for the English dataset results in accuracies over 90%. For the English Dataset the prediction results for classification are mixed, for some sentences they are coherent with the obtained accuracy giving an error only in one/two words for other predictions they are worse.

For the Galician Dataset the accuracy is lower in comparison to the English Dataset but however is overall good achieving over 80% on all sets. The main issue is that upon prediction, the tagging of the words results in an incorrect prediction on all the words.