Source language prediction

In this repository you will find all code and related materials for my thesis about source language prediction.

Reproducing results

1. What you will need

To reproduce my research, you will need the following:

Datasets

- The Europarl corpus [1] (Europarlv7.tar.gz)
- The Books dataset [1] (Books2014-07-30.tar.gz)

Python libraries

- Scikit-learn
- NLTK
- whoswho
- numpy
- shutil
- spaCy (If you want to use parse rules)

[1]: Jörg Tiedemann, 2012, Parallel Data, Tools and Interfaces in OPUS. In Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012)

2. Preparing the data

Background

Preparing the data is done with multiple scripts. The reason for this is twofold.
Firstly, some of these scripts are extremely slow. By using multiple scripts in those cases, it is possible to continue from intermediary results. For example: the find_fragments script has to scan trough multiple GBs of data, and will produce files for all languages afterwards. The split_data script will then actually collect the relevant data from the corpus. If these two scripts would have been put into one, you would have to scan through the whole corpus again if you wanted to work with different languages. Now, you can simply edit the split_data script, and it will collect the languages of interest without scanning again.
Secondly, some scripts have multiple command line options. By putting these scripts in separate files, it is easier to only run the absolute minimum for the parts that you are interested in. Are you, for example, not interested in POS-tags, but in tokens? Feel free to skip all tokenization. By putting these things in separate files, you won't need any libraries that you won't use.

Collecting data

1. The first step in preparing the data is finding the data in the corpus. This is done using the find_fragments script. This script requires two command line arguments (see `find_fragments.py --help`). The first argument is the path to the "raw" folder of your

Europarl corpus. The second argument is the path to the file containing a list of MEPs from the United Kingdom. My version of this file can be found in this repository (meps.txt). Please note that this step takes a very long time, which is why I have put my results of this step in ‘fragment_data.zip’. You should be able to use those results as well.

2. Next, we need to put the data that we are interested in into text files. This is done by running `split_data.py` with the path of the ‘fragment_data’ directory from the previous step, and a path where you want the data to be put.
3. The last step in collecting the data is running the `select_files.py` script. This script expects two arguments. The first argument is the path to your ‘fragment_data’ directory. The second argument is the location where you want the data to be put. If the output directory does not exist, the script will try to create it automatically.

Preprocessing the data

Please note that you can skip this step if you only want to use this system with tokens.

1. Now that we have all data, we want to convert the text files to files containing POS-tags. For this, we can use the `convert_to_pos.py` script. This script expects the path to the directory containing the text files (from the previous step). You can use either NLTKs default tagset, or the universal tagset. (see `convert_to_pos.py --help`)
2. (Optional) I have also included the script that I used to explore parse rules as features. Please note that you need to edit the

classification script to properly use these features as parse rules, as the order of the dependencies will not be considered otherwise.

Preparing the test set

Since the Books dataset is not in the format required for use by my system, we need to prepare that as well. Since this involves some manual moving around of files, you can find my books folder in ‘books.zip’. You can also decide to run `book_to_txt.py`, giving it the path to the Books dataset and the path where you want the files to go. You will then have to manually create the folders for the languages of interest, and put the text files in the correct folder. The language for each of the books can be found in Appendix B of my thesis. You should now be able to run `convert_to_pos.py` on this data as well.

3. Training, cross-validating, and evaluation

Now that we have all data, we can finally train and test the system by running `classify_sk.py`. This script has multiple options, which you can see by running `classify_sk.py --help`. Here are some example usages:

- Train a balanced, part-of-speech based system, and evaluate on the Books dataset.
  `python3 classify_sk.py --balanced --evaluate <PATH TO PREPROCESSED BOOKS> <PATH TO PREPROCESSED DATA>`

- Train an unbalanced, token based system. Don’t evaluate on an external dataset.

```
python3 classify_sk.py --unbalanced --features tokens
<PATH TO PREPROCESSED DATA>
```