One model to rule them all?

Guillaume Wisniewski
guillaume.wisniewski@u-paris.fr

September 2023

- **The objective of this lab is twofold: to make you aware of one of the issues raised by the development of NLP system for languages other than English and to train you in the use of certain tools (typically the automation of shell command launching)**
- **you must submit a report in pdf format ~~September 27th.~~ The report must include for each question either the code allowing to answer it or an answer.** October 9th
- **using google colab might a good idea if you do not have access to a linux computer (I have never installed tensorflow or kenlm on a windows computer and I do everything in my power to never have to do it). Be aware that you will download a lot of data for this lab and you need either a lot of space on your google drive or to work cleverly.**
- **do not hesitate to send me any corrections, comments or suggestions to improve this lab!**

One of the major issues in NLP is to find out if the deep learning models that achieve today state-of-the-art performances can be applied with the same success to all human languages. Indeed, these models have, in the vast majority of cases, only been tested on a very small number of languages generally qualified as ‘resource rich’, which have, from a linguistic point of view, many similarities. If it is therefore not surprising that, until now, a single model (i.e. the same neural network architecture) can be used to achieve very good performances, nothing prove that performance will not drop sharply when these models will be applied to a larger variety of languages.

This lab aims to provide a first answer to this question by verifying, in simple cases, whether there is a link between the performance of a model and certain characteristics of the languages to which it is applied.

In addition to studying this fascinating question (a personal opinion that is not debatable), this lab will also be an opportunity for you to learn how to train a very large number of models (you will have to train several models for each of the 41 languages considered in this lab!) and to highlight the need to automate your experiments.

1 Appetizers: What are we looking for?

We will start by reproducing the results described in [1] that aims at uncovering the relation between the performance of a language model and the morphological richness of a language. Your goal will be to generate a graph similar to the one in Figure 1.

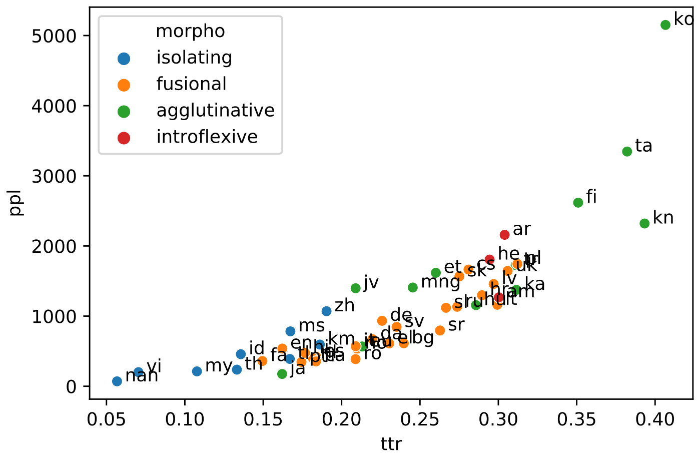

Figure 1: Perplexity of a language model in relation to the TTR for different kind of language and morphological systems.

1. How is the TTR (type-token ratio) defined? Why is this a ‘good’ measure of the morphological complexity? Which other metric could you use to quantify morphological complexity?

2. Why do we consider a language modeling task?

3. How is the perplexity defined? Can we compare perplexity across corpora or languages? Conclude.

4. What conclusions can you draw from Figure 1.

5. Why will we not consider the corpus used in [1]? Why are the FAIR principles[1] a solution to this problem?

2 Main Course

2.1 Generating the corpora

The goal of this part is to create, for each of the language considered, a training set made of 40,000 sentences and a test set made of 3,000 sentences.[2] We will consider the `wiki40b` corpus [2] that can be accessed through the `tensorflow` API. Figure 2 shows how examples from the `wiki40b` corpus can be sampled.[3] The `wiki40b` contains articles extracted from Wikipedia in 41 languages. The dataset has been cleaned up so that it contains only 'content' articles (i.e. pages that describe 'entities') and all the parts with little content (e.g. tables or references) have been removed. The `tensorflow` API allows to access and iterate through Wikipedia articles. Before training LM models, we therefore need to first extract sentences from articles and tokenize them.

```
1 import tensorflow_datasets as tfds
2 from itertools import islice
3
4 ds = tfds.load("wiki40b/fr", split="test", data_dir="gs://tfds-data/datasets")
5 sample = [ex["text"].numpy().decode("utf-8")
6         for ex in islice(ds.shuffle(buffer_size=10_000), 100)]
```

Figure 2: Sampling 100 articles from the `wiki40b` French corpus.

6. What is the role of the `data_dir` parameter in line 4 of Figure 2? What is the meaning of its value?

7. Why do we use the `islice` function in line 6 of Figure 2?

8. Why do we consider the test set (see the `split` parameter)? Is this a good idea?

9. Why do we have to specify a `buffer_size` parameter for the `shuffle` method? How do we choose its value?

10. Why do we have to extract sentences out of Wikipedia articles?

[1]See, for instance, https://www.go-fair.org/fair-principles/

[2]These are the values used in [1]. Feel free to consider larger datasets.

[3]The code provided here works fine on `Google Colab`. Installing the dependencies to run it on your personal computer can be complicated (euphemism) but is a good exercise.

11. Why do we enforce that all train and test sets have the same size?

Detecting sentences boundaries can be tricky especially in a multilingual context in which you do not know most of the languages you are working on. In this work, we will use the Unicode Text Segmentation algorithm to segment text into sentences and words. The `polyglot` library implements this algorithm in Python. This library can be installed (in `Google Colab` or in your local environment) with the following commands:

```
!pip install pyicu
!pip install pycld2
!pip install morfessor
!pip install polyglot
```

The `polyglot` library is also able to tokenize sentences into words. This will be the first tokenization method we consider in this lab.

12. What kind of information `polyglot` is using to identify sentence boundaries and segment sentences into tokens. Comment.
13. For each languages of the corpus extract a train set of 40,000 sentences and a test set of 3,000 sentences. Sentences must be unique (i.e. if there are several occurrences of an identical sentence, they must all be removed but one).
14. Why do we have to remove duplicate sentences?
15. Using `polyglot` tokenize all datasets into words. Save each dataset into a separate text file (one sentence per line) using consistent names to be able to automate the LM estimation (see Section 2.3).

2.2 Extracting morphological information

16. Compute the TTR for all datasets.
17. Knowing that you will have to compute for each language its TTR and its perplexity, what is the best way to store the TTR?
18. Classify each language of the `wiki40b` corpus into one of the following four categories to characterize its morphology: isolating, fusional, introflexive and agglutinative. You can/should use a typological database such as the `WALS`.[4]
19. What structure should you use to store this information? Why?

[4]https://wals.info/

2.3 Training and evaluating language models

In this part of the lab, we will train a language model for each language of the corpus using `kenlm` [3] that provides a very efficient way to estimate a language model with Kneser-Ney. While it weakens the interest of this lab, considering an old, non-neural language model is computationally much more efficient (`kenlm` allows to estimate a language model in a few minutes) and does not require any hyper-parameters tuning.[5]

20. Install `kenlm` using the instructions provided in Figure 3.

21. Why do some shell commands in Figure 3 starts with a `!` and others with a `%`?

```
!git clone https://github.com/kpu/kenlm.git
%cd kenlm
!python setup.py develop
!mkdir -p build
%cd build
!cmake ..
!make -j 4
```

Figure 3: Instructions to install `kenlm` on `Google Colab`.

With `kenlm`, a language model can only be estimated from the command line using the following instructions (in the `build` directory):

```
./bin/lmplz -o 5 < text > text.arpa
```

where `text` is the path to the file containing training data (a text file with one tokenized sentence per line) and `text.arpa` is the path to the filename containing the estimated parameters. The `-o` parameter sets the size of the model (the value of n).

The perplexity of the model can be estimated with the python code of Figure 4.

```python
import kenlm
m = kenlm.Model('something.arpa')
ppl = m.perplexity('a single sentence')
```

Figure 4: Computing the perplexity of *a single sentence* with `kenlm`.

22. Why the python interface of `kenlm` allows you to compute the perplexity of a model but not to estimate its parameters?

[5]But feel free to use any code you want to train a LM and estimate its perplexity.

23. Using a `for`-loop,[6] train a language model for each languages

24. Estimate for each language, the perplexity of the LM and store it into a well-chosen structure.

2.4 Desert

25. Plot the results of the previous two sections to reproduce the result reported in Figure 1.

To ensure that all language have the same vocabulary size, we can consider the BPE tokenization.

26. Install `sentencepiece` and tokenize all datasets using a vocabulary size of 32,000 tokens.

27. Train a language model on these new datasets and compute the new perplexity. What can you conclude?

References

[1] Daniela Gerz et al. "Language Modeling for Morphologically Rich Languages: Character-Aware Modeling for Word-Level Prediction". In: *Transactions of the Association for Computational Linguistics* 6 (2018), pp. 451–465. DOI: `10.1162/tacl_a_00032`. URL: `https://aclanthology.org/Q18-1032`.

[2] Mandy Guo et al. "Wiki-40B: Multilingual Language Model Dataset". English. In: *Proceedings of the 12th Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 2440–2452. ISBN: 979-10-95546-34-4. URL: `https://aclanthology.org/2020.lrec-1.297`.

[3] Kenneth Heafield et al. "Scalable Modified Kneser-Ney Language Model Estimation". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 690–696. URL: `https://www.aclweb.org/anthology/P13-2121`.

[6] We have seen during the second lecture of the course "Introduction au TAL" in L3 how to automate shell commands using a for loop. Variable substitution (see here for an overview) in shell allows to simplify the construction of file names. It is also possible to execute a shell command directly from python (using python `for` instruction).