Multilingual NLP
Lab 1 – One Model to Rule Them All?

AN Ji

Oct 31, 2024

1 Appetizers: What are we looking for?

1. **Definition of TTR** – Gutierrez-Vasques and Mijangos [3]: In quantitative linguistics, the type-token ratio (TTR) is a simple measure of lexical diversity that can also be used for approximation of morphological complexity of languages [5]. It is calculated as the following:

$$TTR = \frac{\#types}{\#tokens} \tag{1}$$

with $\#types$ the total number of **unique** words and $\#tokens$ the total number of words.

Why we consider TTR as a "good" measure – From the above formula, we know that TTR is affected by the numbers of types (i.e. vocabulary size) and tokens (i.e. corpus size). Therefore intuitively, given the same corpus size, languages with a relatively complex inflectional system (e.g. Hungarian) tend to use more lexical forms, thus giving higher TTR; while morphologically poor languages (e.g. Chinese) that show less inflectional capacity rely more on word form repetition, thus giving lower TTR. Kettunen [5] has proven that TTR can be a straightforward yet effective measure to quantify morphological complexity of languages using relatively small corpora. It was shown to be able to "order the languages quite meaningfully in a morphological complexity order" that groups most languages with same kind of languages and clearly separates the most and least morphologically complex ones.

Other measures – There are 7 other measures used for quantifying morphological complexity [1]: Information in word structure (WS), Word entropy (WH), Lemma entropy (LH), Mean size of paradigm (MSP), Inflectional synthesis (IS), Morphological feature entropy (MFH) and Inflection accuracy (IA).

2. **Why we consider a language modeling task** – Because a language modeling task doesn't require annotated data, thus making it much easier to find training data – this is especially crucial to most low-resource languages in the world, because that means we don't need to be some kind of linguistic "specialist" beforehand.

3. **Definition of perplexity** – The perplexity (PPL) is one of the most important metrics in NLP used for evaluating language models [4]. The perplexity of a language model on a test set is the inverse probability of the test set P normalized by the number of tokens N (thus sometimes called the per-word or per-token perplexity). For a test set $W = w_1 w_2 ... w_N$, the PPL of a model is defined as below:

$$\begin{aligned} perplexity(W) &= P(w_1 w_2 ... w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}} \end{aligned} \tag{2}$$

Can we compare perplexity across corpora or languages?

We can't compare perplexity between corpora or languages. PPL is a function of both the corpus and the language model's approximation ability. Given the same corpus W, we can compare different models by perplexity. But if the corpus, or the language varies, then for a corpus/language having more different word forms, the model tends to assign relatively lower probability for each next word w, thus resulting in a higher per-word perplexity, i.e. the corpora/languages themselves constitue a variable that prevents from directly comparing perplexity between them.

4. **Conclusion drawn from Figure 1** – It shows a clear correlation between a model's perplexity on the corpus and the TTR of the language of the corpus: isolating languages (blue) tends to have lower perplexities than most participating languages, followed by fusional languages (orange), then by introflexive languages (red). Agglutinative languages (green) are, however, an exception, as four of them result in the highest perplexities while the others spread out among the other types. We could conclude that language models are better at approximating and predicting languages that have a simpler morphological system.

5. **Why not consider corpus used in Gerz et al.** [2] – Because we no longer have access to the original dataset ("The DS-Web service has been decommissioned [...] on 27 September 2019 affecting services on `people.ds.cam.ac.uk`") and obviously the authors didn't create any redirection from the old URL to a different site.

   Why the FAIR principles is a solution to this problem – If they had followed the FAIR principles, the original data could have been easily findable and accessible to other users who want to reproduce or explore them further.

2 Main course

2.1 Generating the corpora

6. `data_dir` is the location where the datasets are saved (defaults to `~/tensorflow_datasets/`). Hence `gs://tfds-data/datasets` is the directory from which we load the datasets.

7. Here `islice` is used for iterating the 100 Wikipedia articles that Tensorflow is about to pick randomly from the continuously updating buffer of size 10,000.

8. Each of the 41 language datasets has train(90%), dev(5%), test(5%) splits. Here for high-resource languages like French, the test set size (68,004 articles) is enough to provide 43,000 random sentences, and using test set is also faster. However, this strategy would be insufficient for some low-resource languages, especially Filipino, Latvian, with only 1,446 and 1,932 articles in test set, so we had better use train set instead.

9. `buffer_size` defines the data amount (here 10,000) loaded in memory, from which 100 articles will be sampled one by one as the buffer keeps updated with new data from the rest of the original dataset. We need to do this because the whole dataset is too large to process. There should be a trade-off between randomness and memory when choosing `buffer_size`. If memory is a problem, then starting with 1000 to 10,000 as `buffer_size` will be reasonable enough. If not, using a larger `buffer_size` (e.g. equal to the dataset size) can offer better randomness.

10. We choose Wikipedia articles because they share the same descriptive writing style and can cross-linguistically provide parallel/comparable contents about similar topics/entities. In this way we reduce extralinguistic impacts to ensure the datasets are sufficiently homogeneous.

11. We want to see if the model's performance would be affected by morphological features of languages. We equalize the size of train and test sets for each languege because we need to ensure that only the morphological factor itself is the variable that makes a difference, otherwise the model may perform better on high-resource languages since they have more training examples.

12. `polyglot` combines multiple language-specific rules and machine learning algorithms to detect language and identify sentence and token boundaries accordingly: punctuations, whitespaces, word length, context, likelihood of special characters or sequences occuring at the end of sentence, etc.

13. Create train sets (40,000) and test sets (3,000) for 41 languages:

```python
import tensorflow_datasets as tfds
from itertools import islice
from polyglot.text import Text
from polyglot.detect.base import logger as polyglot_logger
polyglot_logger.setLevel ("ERROR")
import os

# os.makedirs("./trains", exist_ok=True)
# os.makedirs("./tests", exist_ok=True)

languages = [
    'ar', 'bg', 'ca', 'cs', 'da', 'de', 'el', 'en', 'es', 'et',
    'fa', 'fi', 'fr', 'he', 'hi', 'hr', 'hu', 'id', 'it', 'ja',
    'ko', 'lt', 'lv', 'ms', 'nl', 'no', 'pl', 'pt', 'ro', 'ru',
    'sk', 'sl', 'sr', 'sv', 'th', 'tl', 'tr', 'uk', 'vi', 'zh-cn', 'zh-tw'
]

# check test set size of each language
test_size = {}
for lang in languages:
    ds = tfds.load(f'wiki40b/{lang}', split='test', data_dir="gs://tfds-data/datasets")
    test_size[lang] = len(ds)

test_size = dict(sorted(test_size.items(), key=lambda lang: lang[1]))
test_size
```

```
{'tl': 1446, 'lv': 1932, 'hi': 2643, 'th': 3114, 'sl': 3341, 'lt': 4683, 'ms': 5235, 'el': 5261, 'hr': 5724, 'sk': 5741, 'et': 6205, 'da': 6219, 'bg': 7289, 'ro': 7870, 'tr': 7890, 'vi': 7942, 'id': 8598, 'he': 9344, 'no': 10588, 'ko': 10802, 'fa': 11262, 'ar': 12271, 'cs': 12984, 'fi': 14179, 'hu': 15258, 'ca': 15568, 'sr': 17997, 'sv': 22291, 'pt': 22693, 'nl': 24776, 'uk': 26581, 'pl': 27987, 'zh-cn': 30355, 'zh-tw': 30670, 'it': 40443, 'ja': 41268, 'es': 48764, 'ru': 51885, 'fr': 68004, 'de': 86594, 'en': 162274}
```

```python
def del_tags(text):
    """remove special markers"""
    text = text.replace("_START_ARTICLE_", " ") \
               .replace("_START_SECTION_", " ") \
               .replace("_START_PARAGRAPH_", " ") \
               .replace("_NEWLINE_", " ")
    return text

def extract_sentences():
    """for each language, extract 43,000 sentences and store in txt files"""
    for lang in languages:
        split = "train" if lang in {'tl','lv','hi','th','sl','lt'} else "test"
        # I choose train sets for the 6 languages that have the smallest test sets (<5k articles)

        ds = tfds.load(f'wiki40b/{lang}',split=split,data_dir="gs://tfds-data/
        datasets")
        sample = [ex["text"].numpy().decode("utf-8") for ex in islice(ds.shuffle(buffer_size=10_000), None)]
        # not setting a slice limit allows to preview the total number of sentences of each language + ensure they are enough

        sentences = {del_tags(sentence) for article in sample for sentence in Text(article).sentences}
```

```
        # print(len(sentences))

        train = list(sentences)[:40_000]
        test = list(sentences)[40_000:43_000]

        with open(f"./trains/train_{lang}.txt", 'w') as f:
            for sentence in train:
                f.write(' '.join(sentence.words) + '\n')

        with open(f"./tests/test_{lang}.txt", "w") as f:
            for sentence in test:
                f.write(' '.join(sentence.words) + '\n')

extract_sentences()
```

14. **Why remove duplicate sentences** – Because we need to ensure the data are representative enough to avoid biasing the model's perplexity on some contents inaccurately.

15. See 13.

2.2 Extracting morphological information

16. Compute the TTR for all datasets:

```
!pip install lexicalrichness

import pandas as pd
from lexicalrichness import LexicalRichness

df = pd.DataFrame(columns=['lang','morpho','ttr'],index=range(41))
df['lang'] = languages
df['morpho'] = ['introflexive' if lang in intr else 'agglutinative' if lang in aggl
    else 'isolating' if lang in isol else 'fusional' for lang in languages]

prefix = '/content/drive/MyDrive/trains/'

ttrs = list()

for lang in languages:
    with open(f"{prefix}train_{lang}.txt", 'r') as f:
        ttrs.append(LexicalRichness(f.read()).ttr)

df['ttr'] = ttrs
```

17. In order to facilitate further calculations and graph plotting, we can create a pandas DataFrame first, with 41 language codes as the first pandas Series (column). As we get informations on the language types, TTR and perplexity scores, we can then create different pandas Series to complete the dataframe.

18. I simply refer to [2] for the typological classification of languages in `wiki40b`:

```
intr = {'ar','he'}  #2
aggl = {'et','fi','hu','ja','ko','tr'}  #6
isol = {'id','ms','th','tl','vi','zh-cn','zh-tw'}    #7
fusi = {'bg','ca','cs','da','de','el','en','es','fa',
        'fr','hi','hr','it','lt','lv','nl','no','pl',
        'pt','ro','ru','sk','sl','sr','sv','uk'}    #26
```

19. See 17.

2.3 Training and evaluating language models

20. Install `kenlm`:

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

21. **Why some shell commands starts with ! and others with %** – `%` is used for magic commands in Jupyter notebooks and specific to IPython environment. The difference between a `%` and a `!` is that the former interacts with the notebook environment, like the `%` here before the two shell commands `cd <some path>`, in which we change the current working directory permanently to `kenlm` and `build` respectively and this will affect the subsequent cells. If we replace them with `!`, then the shell commands will be executed in a subcell and closed after finishing, while the Jupyter environment's working directory remains unchanged.

22. **Why the python interface of kenlm allows you to compute the perplexity of a model but not to estimate its parameters** – Because `kenlm` is designed mainly for providing scoring, rather than training, in a faster way. The training process is separately handled using C++ in command-line tool `lmplz`, and this is much easier and more straightforward than using Python implementation. In this way, the estimated parameters are stored in `.arpa` files for subsequent loading in Python interface to produce perplexity scores.

23. Using a for-loop to train a language model for each language:

```
params = "/content/drive/MyDrive/params/"

%cd '/content/drive/MyDrive/trains/'

for lang in languages:
    !/content/kenlm/build/bin/lmplz -o 5 < 'train_{lang}.txt' > {params}{lang}.arpa
```

24. Estimate for each language the LM's perplexity and store it into the above dataframe:

```
%cd /content/
!pip install https://github.com/kpu/kenlm/archive/master.zip

import kenlm

ppls = list()

for lang in languages:
    m = kenlm.Model(f'{params}{lang}.arpa')
    with open(f'/content/drive/MyDrive/tests/test_{lang}.txt', 'r') as f:
        ppls.append(m.perplexity(f.read()))

df['ppl'] = ppls
df
```

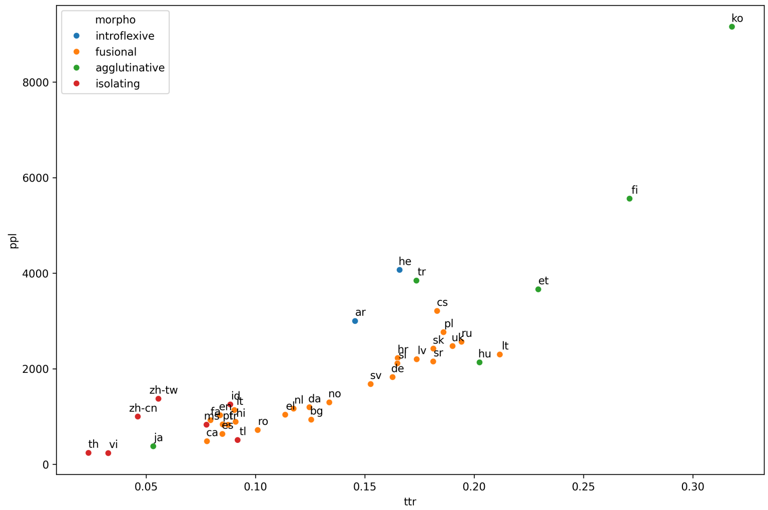
	lang	morpho	ttr	ppl
0	ar	introflexive	0.145565	2997.792404
1	bg	fusional	0.125516	933.967604
2	ca	fusional	0.077845	482.214046
3	cs	fusional	0.183073	3208.965104
4	da	fusional	0.124700	1193.920524
5	de	fusional	0.162709	1824.305017
6	el	fusional	0.113627	1037.915190
7	en	fusional	0.083871	1025.947848
8	es	fusional	0.084901	634.866605

	lang	morpho	ttr	ppl
9	et	agglutinative	0.229329	3661.419786
10	fa	fusional	0.079460	922.699353
11	fi	agglutinative	0.271061	5559.075687
12	fr	fusional	0.087517	826.982826
13	he	introflexive	0.165922	4068.148161
14	hi	fusional	0.090956	889.626151
15	hr	fusional	0.165041	2222.866861
16	hu	agglutinative	0.202471	2132.732830
17	id	isolating	0.088524	1251.588081
18	it	fusional	0.090380	1135.164103
19	ja	agglutinative	0.053298	376.818400
20	ko	agglutinative	0.317872	9156.690666
21	lt	fusional	0.211733	2297.657120
22	lv	fusional	0.173737	2198.258773
23	ms	isolating	0.077639	828.448309
24	nl	fusional	0.117459	1164.744534
25	no	fusional	0.133785	1295.427010
26	pl	fusional	0.185999	2764.540470
27	pt	fusional	0.085023	834.810368
28	ro	fusional	0.101048	716.765832
29	ru	fusional	0.194228	2564.161118
30	sk	fusional	0.181361	2420.749616
31	sl	fusional	0.164920	2111.740707
32	sr	fusional	0.181293	2151.288732
33	sv	fusional	0.152687	1678.658700
34	th	isolating	0.023682	238.791695
35	tl	isolating	0.091857	507.854039
36	tr	agglutinative	0.173608	3843.956043
37	uk	fusional	0.190141	2474.171388
38	vi	isolating	0.032718	233.796116
39	zh-cn	isolating	0.046226	999.280046
40	zh-tw	isolating	0.055664	1371.662722

2.4 Dessert

25. Plot the results:

```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))
sns.scatterplot(data=df, x='ttr', y='ppl',hue='morpho')
for i, lang in enumerate(df['lang']):
    plt.annotate(lang, (df['ttr'][i], df['ppl'][i]), textcoords="offset points",
        xytext=(5,5), ha='center')

plt.savefig('ttr-ppl.png', dpi=300, bbox_inches='tight')
plt.show()
```



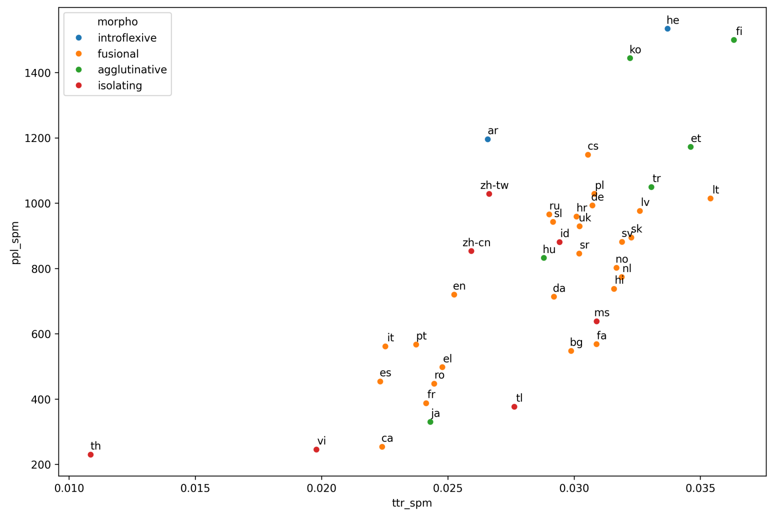
26. Use BPE tokenization, install `sentencepiece` and tokenize all datasets using a vocabulary size

of 32,000 tokens:

```python
!pip install sentencepiece

import sentencepiece as spm

for lang in languages:
    # train sp tokenizers using trains
    spm.SentencePieceTrainer.train(
        input=f'/content/drive/MyDrive/trains/train_{lang}.txt',
        model_prefix=f'/content/drive/MyDrive/spm/models/{lang}_tokenizer',
        vocab_size=32_000,
        model_type='bpe',
    )

    # load trained models
    sp = spm.SentencePieceProcessor()
    sp.load(f'/content/spm/models/{lang}_tokenizer.model')

    # tokenize trains
    with open(f"/content/drive/MyDrive/trains/train_{lang}.txt", 'r') as f_from:
        with open(f"/content/spm/trains/{lang}_tokenized.txt", 'w') as f_to:
            for line in f_from:
                f_to.write(' '.join(sp.encode(line, out_type=str)) + '\n')

    # tokenize tests
    with open(f"/content/drive/MyDrive/tests/test_{lang}.txt", 'r') as f_from:
        with open(f"/content/spm/tests/{lang}_tokenized.txt", 'w') as f_to:
            for line in f_from:
                f_to.write(' '.join(sp.encode(line, out_type=str)) + '\n')
```

27. Train an LM on these new datasets and compute the new ppls. What can be concluded?

```python
%cd '/content/drive/MyDrive/spm/trains/'
for lang in languages:
    !/content/kenlm/build/bin/lmplz -o 5 --discount_fallback < '{lang}_tokenized.txt' > /content/drive/MyDrive/spm/models/{lang}.arpa
    # flag '--discount_fallback' added here because the counts of 1-gram of zh-cn & zh-tw are 'out of range' and considered weird

import kenlm
from lexicalrichness import LexicalRichness

# compute new ttrs
ttrs_spm = list()
for lang in languages:
    with open(f"/content/drive/MyDrive/spm/trains/{lang}_tokenized.txt", 'r') as f:
        ttrs_spm.append(LexicalRichness(f.read()).ttr)
df['ttr_spm'] = ttrs_spm

# compute new ppls
ppls_spm = list()
for lang in languages:
    m = kenlm.Model(f'/content/drive/MyDrive/spm/models/{lang}.arpa')
    with open(f'/content/drive/MyDrive/spm/tests/{lang}_tokenized.txt', 'r') as f:
        ppls_spm.append(m.perplexity(f.read()))
df['ppl_spm'] = ppls_spm

# compute ttr & ppl absolute difference
df['ttr_diff'] = (df['ttr_spm'] - df['ttr']).apply(lambda x: f'{x:+.3f}')
df['ppl_diff'] = (df['ppl_spm'] - df['ppl']).apply(lambda x: f'{x:+.2f}')

df
```



	lang	morpho	ttr_spm	ppl_spm	ttr	ppl	ttr_diff	ppl_diff
0	ar	introflexive	0.026582	1195.418594	0.145565	2997.792404	-0.119	-1802.37
1	bg	fusional	0.029884	547.334715	0.125516	933.967604	-0.096	-386.63
2	ca	fusional	0.022400	253.763639	0.077845	482.214046	-0.055	-228.45
3	cs	fusional	0.030552	1148.071176	0.183073	3208.965104	-0.153	-2060.89
4	da	fusional	0.029204	713.424550	0.124700	1193.920524	-0.095	-480.50
5	de	fusional	0.030727	992.897442	0.162709	1824.305017	-0.132	-831.41
6	el	fusional	0.024784	497.736626	0.113627	1037.915190	-0.089	-540.18
7	en	fusional	0.025253	719.758966	0.083871	1025.947848	-0.059	-306.19
8	es	fusional	0.022324	453.716527	0.084901	634.866605	-0.063	-181.15
9	et	agglutinative	0.034618	1172.173381	0.229329	3661.419786	-0.195	-2489.25
10	fa	fusional	0.030890	568.118511	0.079460	922.699353	-0.049	-354.58
11	fi	agglutinative	0.036331	1499.612843	0.271061	5559.075687	-0.235	-4059.46
12	fr	fusional	0.024142	387.162957	0.087517	826.982826	-0.063	-439.82
13	he	introflexive	0.033704	1534.169082	0.165922	4068.148161	-0.132	-2533.98
14	hi	fusional	0.031583	737.301177	0.090956	889.626151	-0.059	-152.32
15	hr	fusional	0.030097	958.746782	0.165041	2222.866861	-0.135	-1264.12
16	hu	agglutinative	0.028804	832.306617	0.202471	2132.732830	-0.174	-1300.43
17	id	isolating	0.029427	880.538911	0.088524	1251.588081	-0.059	-371.05
18	it	fusional	0.022526	561.204938	0.090380	1135.164103	-0.068	-573.96
19	ja	agglutinative	0.024309	329.997234	0.053298	376.818400	-0.029	-46.82
20	ko	agglutinative	0.032218	1443.982903	0.317872	9156.690666	-0.286	-7712.71
21	lt	fusional	0.035404	1014.243044	0.211733	2297.657120	-0.176	-1283.41
22	lv	fusional	0.032606	975.959910	0.173737	2198.258773	-0.141	-1222.30
23	ms	isolating	0.030894	637.947757	0.077639	828.448309	-0.047	-190.50
24	nl	fusional	0.031889	773.596812	0.117459	1164.744534	-0.086	-391.15
25	no	fusional	0.031680	801.897841	0.133785	1295.427010	-0.102	-493.53
26	pl	fusional	0.030803	1028.391251	0.185999	2764.540470	-0.155	-1736.15
27	pt	fusional	0.023746	566.858929	0.085023	834.810368	-0.061	-267.95
28	ro	fusional	0.024461	446.925682	0.101048	716.765832	-0.077	-269.84
29	ru	fusional	0.029018	965.411439	0.194228	2564.161118	-0.165	-1598.75
30	sk	fusional	0.032268	894.535334	0.181361	2420.749616	-0.149	-1526.21
31	sl	fusional	0.029168	942.497053	0.164920	2111.740707	-0.136	-1169.24
32	sr	fusional	0.030205	845.463018	0.181293	2151.288732	-0.151	-1305.83
33	sv	fusional	0.031900	881.157741	0.152687	1678.658700	-0.121	-797.50
34	th	isolating	0.010856	229.753998	0.023682	238.791695	-0.013	-9.04
35	tl	isolating	0.027636	376.393319	0.091857	507.854039	-0.064	-131.46
36	tr	agglutinative	0.033063	1049.072107	0.173608	3843.956043	-0.141	-2794.88
37	uk	fusional	0.030220	929.084456	0.190141	2474.171388	-0.160	-1545.09
38	vi	isolating	0.019797	245.321639	0.032718	233.796116	-0.013	+11.53
39	zh-cn	isolating	0.025931	853.228303	0.046226	999.280046	-0.020	-146.05
40	zh-tw	isolating	0.026639	1028.204573	0.055664	1371.662722	-0.029	-343.46

Compared to the polyglot tokenization, with the limit of 32,000 vocabulary for all languages, the BPE tokenization splits rare tokens into frequently reused sub-tokens, which reduces the TTR, especially for morphologically complex languages such as agglutinative languages, although the TTR drop of Japanese is relatively not so significant (line 19). In this way the BPE quantitatively narrows the morphological diversity between languages, as well as weakening the correlation between TTR and perplexity (but it can still be roughly observed). One interesting point is that we can see an overall drop of the model's perplexity on any of these languages, except Vietnamese (line 38).

References

[1] Çöltekin, Ç. and Rama, T. (2023). "What do complexity measures measure? Correlating and validating corpus-based measures of morphological complexity". In: *Linguistics Vanguard* 9.s1, pp. 27–43. DOI: 10.1515/lingvan-2021-0007.

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

[3] Gutierrez-Vasques, X. and Mijangos, V. (2019). "Productivity and Predictability for Measuring Morphological Complexity". In: *Entropy* 22.1, p. 48. DOI: 10.3390/e22010048.

[4] Jurafsky, D. and Martin, J. H. (2024). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released August 20, 2024. URL: https://web.stanford.edu/~jurafsky/slp3/.

[5] Kettunen, K. (2014). "Can Type-Token Ratio be Used to Show Morphological Complexity of Languages?" In: *Journal of Quantitative Linguistics* 21.3, pp. 223–245. DOI: 10.1080/09296174.2014.911506.