NLTK Tokenizer for Tigrinya

Cameron Gibson

May 2021

1 Introduction

Tokenization is the process of converting a raw sentence into a sequence of tokens and has a critical impact on the performance of NLP tasks. Packages like Natural Language Tool Kit (NLTK) have traditionally used rule-based parsing methods for tokenization of data. However, NLTK modules only provide rule-based parsing methods for high-resource language. In this paper, a regex-based word tokenizer using NLTK Penn Treebank Tokenizer source code is extended to support the Tigrinya language. The performance of this tokenizer is evaluated against the performance of a Byte-Pair Encoding (BPE) trained tokenizer. Experimental results show returned perfect parsing for Tigrinya can be achieved by adding regular expressions for parsing punctuation markers in the language. However, BPE trained tokenizer showed poor results after tokenization.

As the original proposal for this porject was to develop a morphological analysis tool for Tigrinya using Morfette, I evaluate the coverage of the Horn Morph[1], a Python package that performs morphological analysis, generation, segmentation, as well as grapheme-to-phoneme conversion for various languages of the Horn of Africa including Tigrinya, over the unique tokens in my Tigrinya news corpus.

2 Methods

2.1 Data Collection

Approximately 88k tokens (18k unique tokens) were scraped from a local online Tigrinya news source, the *Haddas Ertrea*, and the Tigrinyan translation of BBC News' online publication. Articles from both news sources were compiled into one data set, where each line of the data set is a randomly sorted sentence from a scraped article. Data sets were then split into train, development (dev), and test sets, where the train set represents 80% of the data, dev set represented another 10% of the data, and the test set represented the final 10% of the data. Training and development sets were used to train our BPE tokenizer, and the same test set was used to evaluate both the results of Tigrinya NLTK tokenizer

[1]https://github.com/hltdi/HornMorpho

and the BPE tokenizer on the test set against a *gold* test set, which is a manually tokenized version of the test set.

2.2 Tokenization

The Penn Treebank tokenizer uses regular expressions to tokenize text, following the implementation is of the tokenizer sed script written by Robert McIntyre.[2] Here, I extended the regular expressions implemented in the Penn Treebank TreebankWordTokenizer class to support Tigrinya punctuation markers. A Hugging Face BPE tokenizer[3] is also trained using the Tigrinya sentence-tokenized news data set for comparison to the NLTK rule-based tokenizer.

3 Evaluation

I evaluated the results of the rule-based tokenizer on the test data set against the results of the BPE tokenizer on the same data set. The NLTK tokenizer performed 100% on both the sentence and word tokenization of the test data set. The BPE tokenizer, however, performed miserably on the sentence and word tokenization of the test data, reporting 30% sentence tokenization accuracy, and 1% on the word tokenization accuracy.

I also evaluated the coverage of the Horn Morph package over uniqued tokens in the Tigrinya news corpus. The Horn Morph contained 54% of the unique tokens in the corpus. However, of those tokens 34% of the tokens are ambiguous, meaning the Horn Morph package contains multiple morphological entries for the word.

4 Conclusion

I find that a rule-based tokenizer implementation still produces the best tokenizations of data. BPE tokenization proved largely ineffective when trained on a smaller data set, reinforcing the notion that the machine learning tokenization methods that back-end large neural machine tranformers like BERT, only perform well on certain tasks when they can be trained on a large data set.

I also find that a reduction in the ambiguity of the morphological entries for Horn Morph would be necessary to develop an effective morphological analyzer with Morfette. I do believe that it might be easy to disambiguate the Horn Morph entries by transferring the data to a different data storing structure like a JSON.[4]

[2]script available at http://www.cis.upenn.edu/ treebank/tokenizer.sed
[3]https://huggingface.co/transformers/tokenizer_summary.htmlbyte-level-bpe
[4]similar to what was done in hm.json

5 Future Work

The most pressing future work that immediately presented itself as a result of this project is the need for tokenization packages that support more languages than just English and other Western Languages. This project was able to show that rule-based tokenizers can provide support for and achieve 100% accuracy on a low-resource language like Tigrinya with small data sets. These results were achieved simply by adding a regex that can identify Tigrinya punctuation. Continued work in this avenue could incorporate FST rule-based tokenization, but at minimum, adding parameters that would allow perhaps a ”language” class that contains a list of regular expressions rules for a given language to be passed as an argument to the tokenizer would be useful.

Machine learning implementations for low-resource languages are a continued topic of discussion in the field. I did think that BPE implementation could possibly be useful for generating a base word vocabularies and morpheme inventories, as we might assume that the constituent symbols that compose morphemes in a language and the morphemes that compose words in a language are likely to symbolically be represented as clusters more frequently than symbol combinations are not morphological significant. However, when deciding how to parse or tokenize a given sequence, frequency distribution is more than likely ill-fit for the task, unless you're working multi-million word data sets. Some rule-based decision-making protocol seems like it will be necessary to improve tokenization accuracy for languages like Tigrinya, where less data is available. Perhaps expansion of packages like Horn Moprh, which include rich linguistic data about the words in the language and the morphology of said word are necessary to begin to create better back-end protocols for tokenization.

As tokenization has been shown to have a direct impact on the performance of NLP models. I am surprised that so little work has been done towards expanding and generalizing tokenization methods for non-western languages, and it's a topic that I think I'll be interested to more actively work on in the future. Likewise, I was originally planning to use Horn Morph as a package for generating feature sets for an implementation of Morfette, so I'd like to do more investigation towards how that might be possible with Horn Morph and so on.