CS 410 Tech Review: Japanese Tokenizers

Sean Kan

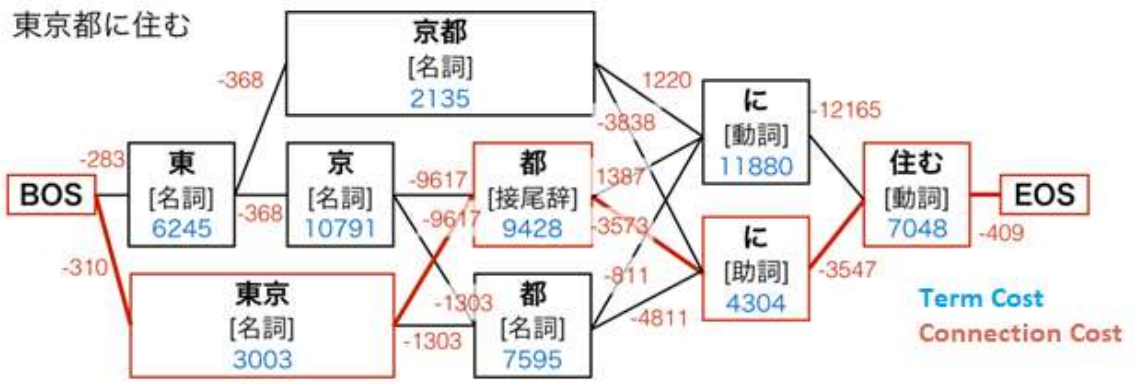
Introduction

Unlike English, Japanese has no spaces between words. The lack of explicit boundaries makes it challenging to perform word segmentation, which is essential for any text information system. During our lecture, it was mentioned that advanced techniques are often needed for predicting the label sequence of such language, which inspired me to conduct further research. This paper first explains the basic concepts behind Japanese tokenization. It then introduces *Sudachi*, a state-of-the-art tokenizer, and lists its advantages over other existing toolkits.

Basic Concepts

Most Japanese tokenizers utilize a term dictionary and a lattice-based model for determining word boundaries. Besides containing a list of known terms in the Japanese language along with their pronunciations and the part of speech, this unique dictionary also includes term and connection costs associated with each word. The term cost is calculated based on word frequency. While the connection cost represents the likelihood that two words will appear in sequential order: the higher the cost, the less likely they are adjacent to each other.

After retrieving this information, the tokenizer will construct a lattice and apply the Viterbi algorithm to locate the minimum cost path. The example below illustrates this concept with the term cost and the term itself as part of the node and connection cost as its path:



The sentence 東京都に住む, which translates to I live in the city of Tokyo, is divided into four tokens(東京,都,に,住む) as a result since connecting these nodes requires the lowest cost possible.

The Sudachi Toolkit

Sudachi is a Japanese tokenizer developed by Works Applications Inc. and the Nara Institute of Science and Technology University as an attempt to replace the archaic but widely-used *MeCab* library that was built 15 years ago. The most prominent feature of this toolkit is that it supports multi-granular tokenization. This allows users to separate words into short, middle, or named entity units based on their needs.

To start using this tool, users would need to pip install two packages with the following commands:

```
pip install sudachipy
pip install sudachidict_core #term-dictionary
```

The following code shows how to tokenize 東京国立博物館 (Tokyo National Museum) with different modes and their respective outcomes:

```
from sudachipy import tokenizer
from sudachipy import dictionary

string = '東京国立博物館'
tokenizer_obj = dictionary.Dictionary().create()

#Short Unit
mode = tokenizer.Tokenizer.SplitMode.A
[m.surface() for m in tokenizer_obj.tokenize(string, mode)]

Results: ['東京','国立','博物','館']

#Middle Unit
mode = tokenizer.Tokenizer.SplitMode.B
[m.surface() for m in tokenizer_obj.tokenize(string, mode)]

Results: ['東京','国立','博物館']

#Named Entity (NE) Unit
mode = tokenizer.Tokenizer.SplitMode.C
[m.surface() for m in tokenizer_obj.tokenize(string, mode)]

Results: ['東京国立博物館']
```

As demonstrated in the example above, the granularity of a token unit can be easily defined by users. Users might prefer shorter units when creating a search retrieval system since it will likely return the highest recall rate out of the three options. The middle unit provides segmentation that is similar to terms written in a conventional dictionary, which could be beneficial to various NLP applications. Meanwhile, the NE units closely resemble real-world entities from a knowledge graph and could be extremely useful for conducting semantic analysis.

In addition, *Sudachi* offers text normalization, which is seldomly featured in an open-source library. As illustrated in the table below, the Japanese language has a complicated writing system that permits character and suffix variations. Fixing this issue is labor intensive as the normalized form of each vocabulary needs to be manually annotated. Moreover, as the language grows and changes over time, new words and variants would need to be continually added to the dictionary. These operations are costly but are made possible because Works Applications Inc., the biggest Japanese vendor of business software for payrolls, has enough resources to dedicate regular maintenance of this toolkit as a part of their long-term business plan.

Type	Example
Kana Suffix Variation (送り違い)	打込む / 打ち込む
Character Type Variation (字種)	かつ丼 / カツ丼
Glyph Variation (異体字)	附属 / 付属
Misspelling (表記誤り)	シミュレーション / シュミレーション

Types of notation variation in Japanese. From Sudachi, a Japanese Tokenizer for Business

Conclusion

Tokenization for a language that does not utilize spaces like Japanese is an extremely challenging task for machines to handle due to word boundary ambiguity. *Sudachi*, a joint project between a system manufacturer and a university in Japan, was created to improve the current situation as it gives users flexibility that is unattainable with other tokenizers. In addition, it features a normalizer that can handle notation variations and regularly updates its dictionary as part of the company's business plan. With all the advantages over its competitors, I strongly believe that *Sudachi* will replace *MeCab* as the industry-standard toolkit for Japanese text processing in the coming years.

References

Takaoka, K., Hisamoto, S., Kawahara, N., Sakamoto, M., Uchida, Y., & Matsumoto, Y. (2018). *Sudachi: a Japanese Tokenizer for Business*.

Kudo, T., Yamamoto, K., and Matsumoto, Y. (2004). *Applying conditional random fields to Japanese morphological analysis.* In Proc. Of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 230-237.

Arabiki, T. (2017, April 23). *日本語形態素解析の裏側を覗く！MeCab はどのように形態素解析しているか*. Cookpad Developers Blog. Retrieved October 28, 2022, from https://techlife.cookpad.com/entry/2016/05/11/170000

Tanakitrungruang, W. (2020, September 6). *How Japanese tokenizers work*. Towards Data Science. Retrieved October 30, 2022, from https://towardsdatascience.com/how-japanese-tokenizers-work-87ab6b256984