An Overview of Japanese Natural Language Processing Tools Through Unsupervised Aspect-Based Sentiment Analysis of Hotel Reviews

Tan Wei Chiong

February 27, 2024

Contents

Abstract

The Japanese language, as with many East Asian languages, contains many semantic differences and nuances compared to English, most of which are cultural. In this paper, we explore the nuances of and tools available in Python for working with a character set that is not Latin-based in natural language processing, through an aspect-based sentiment analysis of Japanese hotel reviews on the Rakuten Travel website. We find that English-based NLP tools are much more well-developed than their Japanese counterparts.

1 Introduction

Since the COVID-19 pandemic, the Japanese consumer market has been slowly becoming more open to purchasing or using products and services from ASEAN countries, especially through e-commerce. Historically, Japanese consumers are also heavily influenced by word-of-mouth recommendations of products and services.[1] This includes reviews as well, which poses an issue when a business owner who does not speak the language gets reviews by users in their native Japanese. While it may be tempting to simply run the reviews through one of the many free online translators, the Japanese language is notorious for its many cultural nuances and indirectness, which may get lost in translation. This led to an impetus by the Japanese government to develop its own culturally-sensitive AI model.[1]

We attempt to attack this issue in two steps:

1. Identifying relevant aspects of Japanese hotel reviews by tokenizing with `natto-py`;
2. Comparing the performance between `oseti` and `asari`, two Japanese language sentiment analyzers.

2 Background and Current Work

The Japanese language has many nuanced complexities that English speakers (or any other language that uses the Latin script) do not have to deal with.

[1]https://the-decoder.com/japan-develops-its-own-culturally-sensitive-language-model/

The most obvious difference is that Japanese uses three different scripts in their writing:

- Hiragana (ひらがな), the basic alphabet of Japanese. This is used for words without any kanji representation, and also as suffixes and prefixes to conjugate verbs;
- Katakana (カタカナ), the script used for non-Japanese loan words, with the same corresponding pronunciations as hiragana;
- Kanji (漢字), which provides semantics for many Japanese words. These originate from the Chinese script, and most have at least two different readings depending on context.

In modern Japanese, the three scripts are often used together, as they play different roles in a piece of text. For example, let us take a sentence found on the Rakuten Travel homepage: "おすすめの宿や特典を見るにはログインしてください" (*o susume no yado ya tokuten wo miru ni wa roguin shite kudasai*), which translates to "Please login to see the recommended hotels and special offers."
The katakana word "ログイン" (*roguin*) comes from the English word "login", while the verb "見る" (*miru*) contains the kanji "見" (*mi*, which means "to see") and the hiragana suffix "る" (*ru*), which indicates that this verb is in its lemmatized form.

The observant reader may have noticed another main difference between the Japanese and Latin scripts. In the English language, words are commonly delimited by spaces. On the other hand, Japanese does not have such a feature, and in fact, the romanized transcription of the sentence "おすすめの宿や特典を見るにはログインしてください" given above is already a testament to how difficult it is for non-speakers of the language (and by extension, a computer) to parse and tokenize words in Japanese.

Because of this difference in structure, while NLP remains largely the same process no matter the language (data cleaning, tokenizing, stemming or lemmatizing, and finally sentiment analysis), the tools used to work with the Japanese language in Python are different from the usual suspects in English – the likes of NLTK and VADER.

Nakayama et al.[2] has also scraped such a dataset, but they had the advantage of labelled aspects in the dataset, which we do not have. This dataset is also not publicly available, and thus we have no choice but to create our own.

3 Methodology

3.1 Data Collection

The dataset we work with is made up of Japanese hotel reviews scraped from the website of Rakuten Travel[2], one of the most popular hotel review sites frequented by Japanese users.

Reviews were scraped using Python's requests library, and in the interest of time, only data from the first two pages of reviews from the first page of hotels per prefecture were obtained. In addition, due to time constraints, only prefectures from the Tohoku region of Japan were considered. These consist of the Akita, Aomori, Fukushima, Iwate, Miyagi, and Yamagata prefectures.

An initial cleaning and filtering is performed during the scrape itself. Specifically, all newline characters `\n` and `\r` are removed before populating the final dataset. All reviews that do not contain ratings are also skipped over.

The final dataset contains 7340 individual reviews and 14 columns. The data dictionary is detailed below.

There are around 10 reviews which are mostly in English. This is filtered out by checking for at least two consecutive alphanumeric strings of length ≥ 4 and separated by a space.

3.2 Japanese Language Tokenization

In regular English NLP, there are a number of standard tokenizers in use. A commonly-used tokenizer is provided by the NLTK library; another comes packaged in Scikit-Learn's CountVectorizer and TfidfVectorizer classes.

All these share the property of tokenizing based on whitespace and punctuation as delimiters. However, as we stated above, Japanese does not come with natural word delimiters outside of punctuation. Instead, a technique known as *lattice-based tokenization*[3] is used. This generates a lattice of all the possible tokens that could surface from a given sentence, along with the probabilities that any given token appears adjacent to another. The *Viterbi algorithm*[4] is then used to find the most probable traversal of the lattice from

[2]`https://travel.rakuten.co.jp/`

[3]`https://towardsdatascience.com/how-japanese-tokenizers-work-87ab6b256984`

[4]See `https://en.wikipedia.org/wiki/Viterbi_algorithm`; this is a use case of statistical parsing.

Column Name	Data Type	Description
review_id	string	Unique identifier of the review
review_time	datetime64[ns]	Timestamp of the review
review_text	string	Actual content of the review
hotel_reply_time	datetime64[ns]	Timestamp of the hotel's reply (if any)
hotel_reply_text	string	Actual content of the hotel's reply (if any)
hotel_name	string	Name of the hotel
prefecture	string	Prefecture that the hotel is located in
overall_score	integer	Overall rating from 1 to 5
service_score	integer	Service rating from 1 to 5
location_score	integer	Location rating from 1 to 5
room_score	integer	Room rating from 1 to 5
amenities_score	integer	Amenities rating from 1 to 5
bathroom_score	integer	Bath and hot springs rating from 1 to 5
food_score	integer	Food and buffet rating from 1 to 5

Table 1: Data dictionary of scraped data

one end to another – in other words, a tokenization of the given sentence.

The dictionary that is used to implement this tokenization is MeCab, named after メカブ (*mekabu*) – a Japanese dish made using the thick leaves near the stem of a species of seaweed – supposedly the developer's favourite food. MeCab is written in C++, but has wrappers for Python. The wrapper which we shall use is natto-py.

Consider the sentence "家族でのんびり過ごせて良かったです。" (*kazoku de nonbiri sugosete yokatta desu*, "I had a pleasant stay with my family.")
The output that natto-py returns takes the following form:

```
家族,名詞,一般,*,*,*,*,家族,カゾク,カゾク
で,助詞,格助詞,一般,*,*,*,で,デ,デ
のんびり,副詞,助詞類接続,*,*,*,*,のんびり,ノンビリ,ノンビリ
過ごせ,動詞,自立,*,*,一段,連用形,過ごせる,スゴセ,スゴセ
て,助詞,接続助詞,*,*,*,*,て,テ,テ
良かっ,形容詞,非自立,*,*,形容詞アウオ段,連用タ接続,良い,ヨカッ,ヨカッ
た,助動詞,*,*,*,特殊タ,基本形,た,タ,タ
です,助動詞,*,*,*,特殊デス,基本形,です,デス,デス
。,記号,句点,*,*,*,*,。,。,。
EOS
```

The important things to take note of is that in each line, the zeroth element is the actual token of the word itself, the first element is the part-of-speech of the token, and the third-last element is the lemma (root or dictionary form) of the token.

As an example, let us take a look at the line corresponding to the token "過ごせ" (*sugose*). It reads

```
過ごせ,動詞,自立,*,*,一段,連用形,過ごせる,スゴセ,スゴセ
```

We see that

- The raw token being detected by natto-py is the zeroth element 過ごせ (*sugose*);
- The part-of-speech of this token is a *verb*, as seen from the first element 動詞 (*doushi*);
- The lemma (or dictionary form) of this token is the third-last element 過ごせる (*sugoseru*). This isn't the actual dictionary form (which is 過ごす, *sugosu*), but the potential form. Regardless, it is close enough for the purposes of natural language processing.

In order to use natto-py as an actual tokenizer with Scikit-Learn tools such as CountVectorizer, we make use of the `tokenizer` parameter in those vectorizers. It allows us to pass a custom tokenizer into the object, and this is what enables Scikit-Learn to work with languages using non-Latin scripts. The key abstraction to remember here is that *tokenizers are functions that take in a document and outputs a list of tokens.* What counts as a "token" is entirely up to us – tokens are simply strings.

In the general-purpose (i.e., what one would normally expect from Scikit-Learn's vectorizers) tokenizer that we build, the particles (助詞), prefixes (接頭詞), conjunctions (接続詞), thanks/greetings (感動詞), and auxillary verbs (助動詞) are excluded, as are punctuation (記号). In addition, the stop words are hard-coded as する, ある, いる, れる, せる, なる, こと, ない, もの, 何, 月, 日, 時, の. Of course, this is non-exhaustive, and would greatly benefit from a more comprehensive review.

We take advantage of this flexibility to also pass in a custom tokenizer that only extracts out the nouns as tokens. Using CountVectorizer, we find the most common nouns that appear in reviews, and hand-pick those that correspond to the six aspects that the hotel reviews fall under.

From there, we use a naïve method of identifying the aspects of a given document. We hard-code a list of keywords that correspond to each aspect, and they are listed as follows:

A third (or fourth, depending on whether one counts the raw MeCab output as a tokenizer) form of tokenization is then performed at the sentence-level, this time aspect-based. A single review is broken up into sentences, which are then classified according to whether they contain any keywords under a specific aspect or not. For example, the review

Aspect	Definition	Keywords
Service	Acts of help towards customer satisfaction	サービス, スタッフ, フロント, チェックイン, 丁寧, 親切, 接客, サーバ
Location	Access and landscape around the hotel	立地, 駅, バス, 近く, 便利, 駐車, コンビニ, 場所
Room	The attributes of the room	部屋, 広い, 宿泊, ベッド, 値段
Amenities	Other facilities in the hotel excluding room and bath	アメニティ, 無料
Bathroom	Bath in the hotel room, or a public bath	風呂, 温泉, 浴場, 露天風呂, 清潔, 湯, トイレ
Food	Morning or evening meals	朝食, 食事, 料理, 夕食, バイキング, メニュー, ご飯, 酒, 飲

Table 2: Definitions and keywords for each aspect

日本酒の飲み比べサーバは良かったです。また、部屋もきれいでスマホ充電など細かな気遣いも良かったです

would be labeled like so:

```
'service': ['日本酒の飲み比べサーバは良かったです'],
'food': ['日本酒の飲み比べサーバは良かったです'],
'room': ['また、部屋もきれいでスマホ充電など細かな気遣いも良かったです']
```

This may be a naïve way to label reviews with their aspects, but as we shall see later, this method already allows us to perform a decent job at sentiment analysis on the aspect level.

3.3 Sentiment Analysis

Sentiment analysis, especially of the aspect-based kind, has historically been mostly well-developed for the English language, with sentiment dictionaries being available as early as 2007. However, the Japanese culture is known to favour a more indirect approach of expressing one's opinion. For instance, the Japanese phrase for expressing the desire that X must be done, "X しなければならない" (X *shi nakereba naranai*) literally translates to "not doing X would be not good," which motivates the need to have sentiment analysis for Japanese specifically.

3.3.1 The oseti Sentiment Analyzer

The oseti library is a dictionary-based sentiment analyzer, which uses a polarity dictionary of words that are already pre-labelled with either positive or negative sentiments.

It works by tokenizing a document at the sentence level, and then giving it a score based on the proportion of "positive"- and "negative"-labelled words present. For example, if one were to analyze the sentence from above:

日本酒の飲み比べサーバは良かったです。また、部屋もきれいでスマホ充電など細かな気遣いも良かったです

The output would be as follows:

```
{'positive': ['良い'], 'negative': [], 'score': 1.0},
{'positive': ['きれい', '良い'], 'negative': [], 'score': 1.0}
```

Essentially, it first tokenizes the document into the sentences "日本酒の飲み比べサーバは良かったです" and "また、部屋もきれいでスマホ充電など細かな気遣いも良かったです". It then picks up the positive word "良かった" (*yokatta*, dictionary form "良い" *ii*, which means "good") from the first sentence, and the positive words "きれい" (*kirei*, pretty) and "良かった" from the second.

This is a very simple approach, and as expected, does not perform too well. This is illustrated in how oseti parses the following (indirect opinionated) sentence:

改善されれば幸いです

which translates to "I hope this improves," referencing the largely negative review that preceded this particular ending sentiment.

oseti would return the output:

```
{'positive': ['改善', '幸い'], 'negative': [], 'score': 1.0}
```

We can see that it incorrectly indicates that this sentence has a completely positive sentiment, purely because it contains the "positive" words "改善" (*kaizen*, improvement) and "幸い" (*saiwai*, for the best). Clearly, there could be some improvement here.

3.3.2 The asari Sentiment Analyzer

Unlike oseti, the asari sentiment analyzer is not purely dictionary-based. Instead, it uses a pipeline consisting of Scikit-Learn's TfidfVectorizer and a linear support vector machine, though what data it trained on was not mentioned. It was written in 2019 after the author found that most publicly available sentiment analyzers at that time were written for English, and the few that were capable of processing Japanese text were too costly to maintain.

According to the metrics given by the author of asari, it performed just slightly worse than the then-new BERT model of that time, which one could either interpret as a testament to the power of asari, or an example of how much better BERT has become since then.

It works by predicting a probability of an input document having positive sentiment, and returning that probability along with its complement $(1 - P(\text{positive}))$ as a negative. For our use case, we take a net sentiment (positive − negative) as the sentiment score for a given input.

For example, using asari to analyze the sentence that was incorrectly classified by oseti above:

改善されれば幸いです

The output from asari would be

```
{'text': '改善されれば幸いです',
'top_class': 'negative',
'classes':
[{'class_name': 'negative', 'confidence': 0.8300549387931824},
{'class_name': 'positive', 'confidence': 0.16994500160217285}]}
```

It correctly predicts the sentiment of that sentence to be negative, with an 83% confidence level. Thus, the net score we shall give this sentence would be -0.660109937191.

At this point, a single example would not be enough to quantitatively illustrate which of the two sentiment analyzers perform better. This is difficult for us due to the lack of labelled data. However, we can work around this issue by using the sentiments to give a rating from 1 to 5 for each aspect of the review, and compare that to the actual ratings given by guests.

4 Results

All the (aspect-based) sentiment scores that were populated from oseti and asari are in the range $[-1, 1]$. If we wanted to convert them to a rating within the range $[1, 5]$, a simple linear mapping would suffice. The resulting number would then be rounded off to the nearest integer. Due to the undesirable behaviour of numpy's `round()` function (which is biased towards even numbers), we instead use the mapping

$$x \mapsto \lceil 2x + 2.5 \rceil$$

This allows us to directly compare how well the sentiments match with the given ratings. There are two types of metric that could be used to quantify how well the sentiment analyzers perform

1. Scikit-Learn's `classification_report()`, which gives us all the relevant metrics when viewing this as a comparison of two classifiers;

	Precision	Recall	F1-score	Support
1	0.30	0.37	0.33	86
2	0.13	0.16	0.14	139
3	0.32	0.13	0.19	568
4	0.47	0.14	0.21	2186
5	0.54	0.93	0.68	2589
accuracy			0.51	5568
macro avg	0.35	0.35	0.31	5568
weighted avg	0.48	0.51	0.43	5568

Table 3: Overall classification report for asari

	Precision	Recall	F1-score	Support
1	0.03	0.03	0.03	86
2	0.08	0.14	0.11	139
3	0.16	0.37	0.22	568
4	0.36	0.42	0.39	2186
5	0.51	0.26	0.35	2589
accuracy			0.33	5568
macro avg	0.23	0.25	0.22	5568
weighted avg	0.40	0.33	0.34	5568

Table 4: Overall classification report for oseti

	Precision	Recall	F1-score	Support
1	0.23	0.06	0.10	78
2	0.26	0.17	0.21	144
3	0.21	0.85	0.33	952
4	0.28	0.07	0.11	1894
5	0.69	0.29	0.41	2500
accuracy			0.31	5568
macro avg	0.33	0.29	0.23	5568
weighted avg	0.45	0.31	0.29	5568

Table 5: Service classification report for asari

	Precision	Recall	F1-score	Support
1	0.12	0.10	0.11	78
2	0.18	0.08	0.11	144
3	0.20	0.85	0.33	952
4	0.20	0.04	0.07	1894
5	0.65	0.27	0.38	2500
accuracy			0.28	5568
macro avg	0.27	0.27	0.20	5568
weighted avg	0.40	0.28	0.25	5568

Table 6: Service classification report for oseti

	Precision	Recall	F1-score	Support
1	0.05	0.05	0.05	19
2	0.06	0.03	0.04	102
3	0.16	0.77	0.26	833
4	0.33	0.06	0.11	2068
5	0.59	0.26	0.36	2546
accuracy			0.26	5568
macro avg	0.24	0.23	0.16	5568
weighted avg	0.42	0.26	0.24	5568

Table 7: Location classification report for asari

	Precision	Recall	F1-score	Support
1	0.01	0.05	0.01	19
2	0.08	0.05	0.06	102
3	0.16	0.82	0.27	833
4	0.32	0.04	0.07	2068
5	0.60	0.20	0.30	2546
accuracy			0.23	5568
macro avg	0.23	0.23	0.14	5568
weighted avg	0.42	0.23	0.20	5568

Table 8: Location classification report for oseti

	Precision	Recall	F1-score	Support
1	0.17	0.03	0.06	87
2	0.20	0.05	0.08	235
3	0.19	0.58	0.29	924
4	0.32	0.12	0.18	1989
5	0.55	0.45	0.49	2333
accuracy			0.33	5568
macro avg	0.29	0.25	0.22	5568
weighted avg	0.39	0.33	0.32	5568

Table 9: Room classification report for asari

	Precision	Recall	F1-score	Support
1	0.07	0.13	0.09	87
2	0.21	0.12	0.15	235
3	0.19	0.73	0.30	924
4	0.30	0.10	0.15	1989
5	0.56	0.25	0.35	2333
accuracy			0.27	5568
macro avg	0.26	0.26	0.21	5568
weighted avg	0.38	0.27	0.26	5568

Table 10: Room classification report for oseti

	Precision	Recall	F1-score	Support
1	0.10	0.02	0.03	65
2	0.20	0.03	0.05	218
3	0.25	0.94	0.39	1360
4	0.37	0.02	0.04	1976
5	0.49	0.07	0.13	1949
accuracy			0.26	5568
macro avg	0.28	0.22	0.13	5568
weighted avg	0.37	0.26	0.16	5568

Table 11: Amenities classification report for asari

	Precision	Recall	F1-score	Support
1	0.00	0.00	0.00	65
2	0.22	0.01	0.02	218
3	0.25	0.95	0.40	1360
4	0.23	0.01	0.02	1976
5	0.50	0.07	0.13	1949
accuracy			0.26	5568
macro avg	0.24	0.21	0.11	5568
weighted avg	0.33	0.26	0.15	5568

Table 12: Amenities classification report for oseti

	Precision	Recall	F1-score	Support
1	0.11	0.06	0.08	69
2	0.19	0.07	0.10	206
3	0.29	0.80	0.43	1153
4	0.31	0.10	0.15	1840
5	0.62	0.48	0.54	2300
accuracy			0.40	5568
macro avg	0.31	0.30	0.26	5568
weighted avg	0.43	0.40	0.37	5568

Table 13: Bathroom classification report for asari

	Precision	Recall	F1-score	Support
1	0.07	0.16	0.10	69
2	0.13	0.05	0.08	206
3	0.28	0.85	0.42	1153
4	0.26	0.07	0.12	1840
5	0.62	0.35	0.45	2300
accuracy			0.35	5568
macro avg	0.27	0.30	0.23	5568
weighted avg	0.41	0.35	0.31	5568

Table 14: Bathroom classification report for oseti

	Precision	Recall	F1-score	Support
1	0.28	0.08	0.12	106
2	0.09	0.05	0.06	192
3	0.24	0.65	0.35	827
4	0.30	0.14	0.19	1816
5	0.64	0.57	0.61	2627
accuracy			0.42	5568
macro avg	0.31	0.30	0.27	5568
weighted avg	0.44	0.42	0.40	5568

Table 15: Food classification report for asari

	Precision	Recall	F1-score	Support
1	0.09	0.13	0.11	106
2	0.10	0.06	0.08	192
3	0.22	0.72	0.34	827
4	0.24	0.10	0.15	1816
5	0.62	0.42	0.50	2627
accuracy			0.34	5568
macro avg	0.25	0.29	0.23	5568
weighted avg	0.41	0.34	0.34	5568

Table 16: Food classification report for oseti

2. Scikit-Learn's `mean_absolute_error()`, which serves as a measure of how close the sentiment analyzer got to the actual rating.

It is clear that even though asari performs somewhat better than oseti in terms of the classification metrics, they are both objectively quite bad in predicting the exact rating that customers give, as arbitrary as it may be.

On the other hand, the mean absolute error may be a better metric to judge our classifiers, as from a business perspective, it would not be too bad if the predicted scores are close to the actual ones. The MAE for each aspect and analyzer are given as follows:

	oseti	asari
Overall	0.836027	0.600395
Service	1.054956	1.008441
Location	1.183908	1.095725
Room	1.072557	0.953125
Amenities	1.086745	1.084590
Bathroom	0.943426	0.839260
Food	0.947198	0.795617

Table 17: Mean absolute errors for each aspect and analyzer

Though the classification report does not show good results, we see from the MAE analysis that asari does predict a closer rating than oseti on average.

5 Limitations and Future Work

Many limitations of this project derive from the fact that all this was done within the span of around two weeks. Therefore, given more time, the following would be good to have.

- *Collecting more data.* Currently, we only have 40 reviews each from the first page of hotels from 6 prefectures. Obtaining a more complete dataset from all 47 prefectures of Japan would allow for a much better analysis of the aspects present in the data, which would in turn lead to more accurate aspect-based sentiment analysis.

- *Proper labelling of the aspects present in each review.* As stated in the introduction, Nakayama et al. has a similar dataset, which took 43 native Japanese people and around 2 months to annotate, on top of an initial crowdsourced annotation of the unlabelled dataset. On the other hand, the approach we took in this project was rather programmatic and simple in nature – a far cry from the nuanced parsing by human readers of the reviews. This could therefore be improved either by manually going in to annotate everything and then resolving conflicts, or by figuring out a better aspect detection algorithm for Japanese.

- *Using a neural network for aspect-based sentiment detection.* This two-week effort was mostly focused on pushing out a minimal working proof-of-concept. As a result, in the interest of time, sentiment analyzers that do not take long to run were selected for comparison. However, better tokenizers and sentiment analyzers that use neural networks (e.g., BERT models) do exist, though there are few compared to those built for English NLP. We wish to explore that in the future under a more generous schedule.

- *A comparison with translated reviews.* From the perspective of a non-Japanese business owner, a solution to the issue of having reviews that consist of Japanese text would be to simply pass it through one of the many online translators in order to read it. This was very briefly explored with `deep-translator`'s GoogleTranslator as the translator, and VADER as the sentiment analyzer. What was briefly seen showed great promise, though that mainly shows how advanced Google's translation architecture has developed over the years.

6 Conclusion

In this paper, we presented a brief analysis of how one would go about working with non-Latin (mainly Japanese / East Asian) scripts in Python, and a comparison of the available tools that are developed specifically to deal with Japanese-specific language processing issues and nuances. We hope to revisit the different avenues of development outlined in the previous section and explore more tools for non-English NLP (perhaps even developing one ourselves), time permitting.

7 Acknowledgements

The author would like to thank General Assembly's class of SG-DSI-41 – in particular, Waseem Sheriff and Rustam Shariq Mujtaba. This endeavour would not have been possible without the support of the wonderful people in SG-DSI-41, and many lessons have been learned along the way.

References

[1] The Japanese consumer market. `https://www.asean.or.jp/en/wp-content/uploads/sites/3/Japan_consumer-_market.pdf`. Accessed: 2024-02-20.

[2] Yuki Nakayama, Koji Murakami, Gautam Kumar, Sudha Bhingardive, and Ikuko Hardaway. A large-scale Japanese dataset for aspect-based sentiment analysis. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 7014–7021, Marseille, France, 06 2022. European Language Resources Association.