Japanese WSD

Justin Veyna jcveyna@uci.edu
https://github.com/JustinVeyna/JapaneseWSD
University of California, Irvine

1 Abstract:

This document depicts an approach to Word Sense Disambiguation (WSD) for Japanese text. The document also covers difficulties that are not present in WSD for English text. In this paper I only uses data that is open to all for research purposes.

2 Introduction:

WSD Word sense disambiguation is a process of taking a word within a piece of text and discerning the meaning of the word amongst its potentially many meanings. For example the word 「力」(Power) has multiple meanings including: a powerful effect or influence, forceful exertion, the property of being physically or mentally strong, and many more. So, WSD is finding out which meaning, or sense, 「力」refers to in the sentence 「彼は力を持っています。」(He has power.).

Applications WSD has important applications in various fields including search engines and smart assistants. If a user searches a phrase or word, it is important to understand which meaning of the word the user is referring to in order to achieve more relevant results. Likewise a smart assistant needs to be able to correctly discern the user's commands even if they use an ambiguous word or phrase.

Solutions Some preexisting solutions include clustering, always picking the most popular sense, wordnet walk, and using an SVM classifier. Clustering creates a cluster for each sense and then creates a vector for the word and question and searches for the closest sense. My approach is similar to clustering in that it creates an average vector for each sense and a vector for the word in question and searches for the closest sense.

3 Problem setup:

3.1 Japanese innate problems:

Using an English-Japanese linked word-net I was able to get a list of senses for each Japanese word. From there given a sentence, and a word in the sentence, the algorithm has to discern which sense the word refers to. There are a few additional problems that Japanese faces that English does not.

Tokenization Japanese words are not split by spaces. So, unlike English, tokenization is not trivial. Furthermore, characters can have meanings by themselves, and when combined with certain other characters can change in meaning. For example 「出」meaning "leave" and 「口」meaning "mouth", when put together, 「出口」, means "exit". In this case the meaning of the combined characters can be discerned by the meaning of the individual characters put together, but this is not always the case. For example, in the word 「お土産」, 「お」is a politeness modifier, 「土」means dirt, and 「産」means produce, but when put together it means “souvenir.” Taken a step further, characters in Japanese are made of one or more radicals, which have their own meaning as well. For example the character 「森」 meaning “forest” is made up of three 「木」 radicals, meaning “tree.” But, a character’s meaning cannot necessarily be guessed through just knowing the radicals and their meanings. If you take the radical meaning evening 「夕」 twice then you get the character 「多」 meaning many. Overall, words can be tokenized at varying levels while still retaining meaning.

Sense Count Japanese words tend to have more senses than English words, leading to naturally lower scores for WSD on Japanese texts vs. WSD on English texts.

Text Availability the amount of available Japanese text on the internet is much less than the amount of English text on the internet. This limited the number of corpora I could find that were free to use. But this being said there is still a plethora of Japanese text online.

4 solution:

My solution is to create sense-vectors for each sense, tokenize the given sentence into words, then for the desired word look up the senses for that word and decide on a sense.

Sense Vectors For this approach it is important to create a sense vector for each sense. To do this I first look up all words with that sense. I then assign the sense vector for that sense to be the average word2vec vector of all words with that sense.

Tokenization Amongst all of the ways to tokenize Japanese, I chose to tokenize then at a word level because it is the most stable while still retaining a manageable set size. One downside to this method is that for proverbs such as 「一石二鳥」 (2 birds one stone) it fails to capture the true meaning. But proverbs like this are uncommon. I opted to use the tokenizer library (TinyTokenizer).

Context Vector Much in the same way as for sense vectors, I took the average of all the word2vec vectors for each word in the sentence to create a context vector. This context vector contains the rough meaning/context for the whole sentence.

$$\bar{x} = \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)$$

Sense Scoring I then looked up the senses for the word in the word-net and pulled up the pre-calculated sense-vector for each sense. Using the sense-vectors for the word I ranked them by descending distance to the context vector. This rank represents the sense that the algorithm predicts is correct.

Distance Formula Since magnitude makes a difference in the meaning of the word: using Euclidean distance makes more sense since it preserves the magnitudes. Cosine Difference on the other hand ignores magnitude and just finds an angle.

$$\sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}.$$

5 experiments:

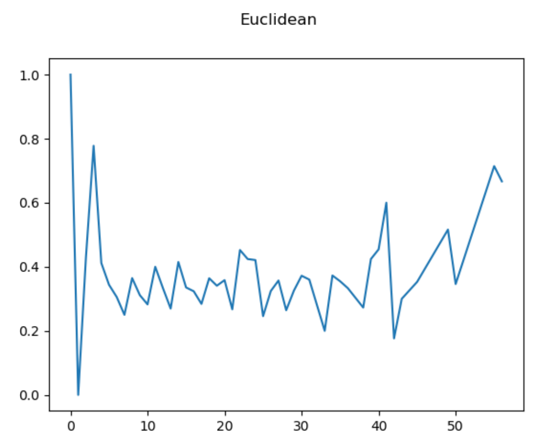
Data The corpus I used contained 55754 words labeled with at least one sense. I used this labeled data as test data. Each word has an average of 5.07994438651373 senses per word.

Sense Ignorance One issue I encountered is sense ignorance, the fact that some of the labeled senses were not listed as a sense for the word. The missing senses were usually in related words. For example 「ある」 (to exist) might have been labeled for a sense in 「である」 (to be). Sense ignorance affected roughly 25% of the sense labeled words.

I tried to address this issue by including words that contained the word that was being looked at, but this resulted in the average sense per word to jump up to nearly 200 senses per word. This both slowed down computation time and even resulted in a lower accuracy of 25.69%, and an MRR of 0.3923

I decided to ignore sense ignorance and lose the potential 25% accuracy.

Sentence Length I also tested to see if sentence length had a significant impact on accuracy to see if using sentences for the context vector is sufficient of perhaps even too wide of a frame. The results were noisy due to the small size of my corpus, but it was enough to show that there is no significant difference between the accuracy for longer and shorter sentences.



Random With about 5 senses per word I expected random to get about 20% accuracy and after testing it, got 20.13% accuracy.

Cosign Difference Using cosign difference as the difference formula for comparing sense vectors to the context vector got an accuracy of 32.92%

Euclidean Distance Using Euclidean distance as the difference formula for comparing sense vectors to the context vector got an accuracy of 35.98%

Euclidean distance (context vector for the whole paragraph) When using a context vector using the whole paragraph versus the sentence containing the word I found that the accuracy changed less than .01%.

Distance Formula	Accuracy
Euclidean Distance (sentence)	35.979%
Euclidean Distance (paragraph)	35.976%
Cosine Difference (sentence)	32.916%
Random	19.685%

Other papers were able to get percentages above 95% accuracy, but the gap is due to my algorithm's reliance on the word net, inconsistencies in tokenization and how these affects the calculations. As mentioned earlier, Word Sense Ignorance played a major role in reducing accuracy, making my model unable to guess about a quarter of the senses correctly. Furthermore, approximately 5% of the words are mislabeled. Lastly, not all words have senses associated with them and even if they did, the tokenizer may have resulted in a different word being generated. For these words I decided not to include them in the accuracy calculation.

5.2 Example Inputs

In the sentence「円周率は何ですか？」(What is pi?) the character「円」 can mean both circle and Japanese yen. In this case the algorithm incorrectly identifies the character「円」as referring to the Japanese yen

In the sentence 「私の声を聞こえますか？」(Can you hear my voice?), the algorithm correctly identifies the word 「声」, meaning voice, to mean a medium of speech/the ability to speak over the linguistics term and mechanically generated sounds.

6 Related work

Shirai, Kiyoaki and Nakamura, Makoto attempted a similar problem to WSD for Japanese, but they further specialized their work towards identifying new, unlabeled senses for words. They ensembled together SVM and clustering. While they had limited success with finding new word senses, they were able to do WSD to a high accuracy.

7 Conclusion

One of the biggest issues I faced was the rigidness of tokenization. If the tokenizer tokenized words into a different form than that in the word net or in the word2vec, then the word was ignored as there is no information on it. In the future, I would like to use the tokenizer for all steps of the process instead of relying on data from various different sources. This is especially important for Japanese as tokenization can be ambiguous. Due to the discrepancy in tokenization and my reliance on the corpus, the way I measure accuracy is for too different than that of other papers. This resulted in work in being hard to compare to that of others.

8 References

Shirai, Kiyoaki & Nakamura, Makoto. (2010). JAIST: Clustering and classification based approaches for Japanese WSD. 379-382.

Japanese Wordnet (vXX) © 2009-2011 NICT, 2012-2015 Francis Bond and 2016-2017 Francis Bond, Takayuki Kuribayashi