Programming Assignment 2: Word Sense Disambiguation

Hanying Shao

260851798

Abstract

In this project, I investigated the performance of 4 different methods: Lesk Algorithm, Cosine Similarity with Word2vec, Lesk Algorithm with inverse frequency and Baseline on the Word Sense Disambiguation (WSD) problem. From the experiment, I found that the Baseline Algorithm generally resulted in better accuracy on the given dataset. I implemented 4 classes to disambiguate given words and tried to improve the accuracy by adding an inverse frequency value to the lesk algorithm. I also did some researches online and found the cosine similarity algorithm to test the similarity of two given documents with Word2vec which uses a neural network model to learn word associations from a large corpus of text. I loaded a word2vec from a pre-trained Google News corpus word vector model. Meanwhile, I also tried to add a bootstrap sampling to the improved Lesk Algorithm. The analysis of results is carried out based on the accuracy of different models. A conclusion and possible improvement ideas were driven as well.

Problem Settings

The dataset used for the experiment is 1644 ambiguous words from some sentences. First, I use the methods implemented by Prof. Cheung in loader file to load the lemmas and contexts from "multilingual-all-words.en.xml" and target keys "wordnet.en.key." (Cheung, 2020)Then, I split the datasets into two subsets, *dev* and *test* having 194 and 1450 elements respectively.

First, I implement the baseline algorithm. With the built-in wordnet.sysnet(lemma) function, we can get a list of possible synsets already ordered by the frequency. Thus, I just use the first one in the list to be the predicted value and then transform the synset into the key to match the given target values. The accuracy is calculated by dividing the correct predictions by the total number. Baseline Algorithm gives a surprisingly high accuracy in the given dataset.

Secondly, I implemented the Lesk Algorithm using the built-in *lesk* function from *nltk* package. Since the Lesk Algorithm uses contexts in which the ambiguous word is present to determine the meaning of the word. I use some preprocessing methods such as lemmatization, stemming and removing stopwords to clean the context sentences before passing it to the *lesk* function.

First, I only used the first lemma of the predicted synset and compare its key value to the target value. I got about 20% accuracy for Lesk Algorithm and 50% accuracy for Baseline Algorithm. After I check the outputs carefully, I noticed that some output synsets contain the correct lemma which is not placed as the first one. Thus, I changed my function to output a list of keys of all lemmas of output synset and as the target values are also a list of keys, I check if there exists an intersection of two lists. The accuracy of models increases significantly after correction.

```
built-in lesk accuracy on dev_instances:  0.3917525773195876
baseline accuracy on dev_instances:  0.6752577319587629
built-in lesk accuracy on test_instances:  0.30551724137931036
baseline accuracy on test_instances:  0.623448275862069
```

Figure 1: output results for lesk and baseline algorithms

After some doing researches, I found that we can compute weighted overlap using the inverse document frequency, *idf* from a paper (Ahmed H. Aliwy, Ayad R. Abbas). *idf* measures the importance of word in a corpus by reducing the power of common words appearing in every sentence, such as this, the, etc. the formula is: $idf_i = log\frac{N}{n_i}$ where N is the number of document in the collection and n_i is number of documents containing the word w_i. I use the inverse frequency to get a dictionary of word with its *idf* value for all definition of a lemma and use it to calculate the overlap value and the best overlap value corresponds to the most possible definition of the word. I resample the dataset into 5 subsets and perform the test on the subsets to bootstrap. The accuracy is significantly higher using this method.

```
My lesk accuracy on dev_instances:  0.5979381443298969
My lesk accuracy on test_instances:  0.5674482758620689
```

Figure 2: output results for improved lesk algorithms

Finally, I found Cosine Similarity Algorithm collaborate with Word2Vec to disambiguate words from an article on *future internet*. (Yang, 2019) Cosine similarity is a metric used to measure the similarity of two sentences irrespective of their size. It measures the cosine of the angle between two vectors in a multi-dimensional space. (Karani, 2018) Word2Vec is one of the most popular technique to learn word embeddings using shallow neural network which are binary representations of words. I used a pretrained model *GoogleNews-vectors-negative300.bin* from Amazon. The whole is large, so to reduce the processing time, I set a limit of 2000. The idea is to find all the possible definition of a words including its hypernyms, hyponyms, etc. and transform it into vectors, then calculate the cosine similarity of the definitions with the sentence. The synsets are ordered by the cosine similarity scores. I only output synset having the highest score to be my output. The synset could have multiple lemmas. I output the list of all keys of the synset and check if there exists an intersection with target values. This method has a relatively high accuracy around 53%.

```
cosine similarity with word2vec accuray on dev_instances 0.5360824742268041
cosine similarity with word2vec accuray on test_instances 0.5393103448275862
```

Figure 3: output results for cosine similarity algorithms

I tried to increase the limit of Word2Vec model to see if it can improve the accuracy. However, when I increase the limit from 2000 to 5000, there is not a significant change on the accuracy of the model. I think that it is because given dataset is not very large so, we do not need too many word vectors from pretrained model.

Conclusion

From the results, we can see that the built-in lesk algorithm has a worst result. However, when we improved the lesk function with the inverse document frequency, the accuracy increases significantly which implies that the common words in the definitions of lemmas have a great impact while calculation of overlap value. If we do not consider the *idf*, the accuracy is poor. The cosine similarity is also a very interesting algorithm to test on the similarity of two sentences. Instead of calculating the overlap of sentences, it transforms the sentence into vectors using a word-embedding approach, Word2vec and determine the similarity by computing the vector angle. The accuracy of this method is also slightly lower than the improved lesk algorithm. However, the dataset we used is not large enough to determine which method has a better performance. Finally, the baseline algorithm has surprisingly a highest accuracy in the experiment. However, as mentioned above, we cannot conclude that baseline algorithm outperforms all other algorithms because the dataset is limited and after checking the sentences, I found that some sentences are very short and ambiguous. It could also be the reason why lesk algorithm fail to disambiguate the word in the sentence. Thus, the most frequent sense of word could just happen to be the right answer.

Nonetheless, the accuracy of the algorithms is generally below 60% which means that we can further improve the algorithm to improve the model. For example, adding a regularization term to the formula to see if it can reduce the weights of unimportant words. In addition, we need to conduct an experiment on a larger dataset to make the results more significant in various context.

In brief, from the experiment, we experiment on different method to disambiguate words in a sentence. The baseline algorithm has a best accuracy while built-in lesk function gives a lowest accuracy. However, it is not possible to analyze the performance of the algorithm because the dataset is limited.

References

Ahmed H. Aliwy, Ayad R. Abbas. (n.d.). *IMPROVEMENT WSD DICTIONARY USING ANNOTATED CORPUS AND TESTING IT WITH SIMPLIFIED LESK ALGORITHM.* University of Technology, Baghdad .

Cheung, J. (2020). *Loader.py and Class Notes.* November.

Karani, D. (2018). Introduction to Word Embedding and Word2Vec. *Towards Data Science*.

Yang, W. (2019, May 12). Word Sense Disambiguation Using Cosine Similarity Collaborates with Word2vec and WordNet. *Future Internet*, p. 16.