Distributional Lesk: Effective Knowledge-Based Word Sense Disambiguation

Natural Language Processing - Computing Meaning
Final Report
Ulrich Heid, Christian Wartena, Johannes Schäfer
Submitted by - Simran Kaur 311443
kaur@uni-hildesheim.de

Abstract

In this Report, we will be evaluating a simple Word Sense Disambiguation method that uses the same idea as the classical Lesk algorithm to find the sense of a word in a specific context. As the Lesk algorithm takes into account the overlap between the context of a word and the gloss of its senses, a similar idea has been used here with a change in the way this overlap will be handled. We use the embeddings, instead of counting the number of words that overlap, to see the similarities between the senses of a word and its context. The model is evaluated on the three datasets SemCor3.0, Senseval2, and Senseval3 for the English language and has shown to perform better than both the Baseline methods mentioned in the paper. The method is further extended to see the effects on the performance by removing stop words, using other pre-trained word embeddings such as Word2Vec, and, using sentence embeddings such as Flair and SBERT to report the accuracy.

1 Introduction

Word Sense Disambiguation (WSD) refers to finding the correct meaning of a word in a context. A word in a sentence can have multiple meanings and finding the right meaning with reference to the context has been an important problem in natural language processing. So far, the best performing WSD systems are based on supervised learning [1], [2], [3], [4], [5] which needs manually annotated target points to train the model. The performance of such supervised systems can be good in some domains, but they might not be extended to other domains in some cases [6].

In contrast, Knowledge-based systems do not require manually annotated target points, as well as they can be extended to new domains easily [7]. To find the meaning of a word in a context, two things are required (1) the definition of each of the possible senses of the word and (2) the context of the word. One example of such a system is the Lesk algorithm [8], which considers the overlap between the definition of a sense and the words in the context to find the meaning of the ambiguous word.

In this paper we will go through a knowledge-based WSD method based on the Lesk algorithm that also considers the overlap between the definition of senses and the words in the context of the target word is proposed. Instead of using the intersection between the set of gloss definitions and the context set, we take the word and sense embeddings to compute the similarity between the sense of the word and its context. The main disadvantage of the supervised methods was that they require a large amount of labeled data, which has been overcome here as the knowledge-based systems just require a sense inventory such as WordNet, and the unlabelled data to work upon. For all the other languages whose sense inventory is available, this method can be easily extended for them.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

2 Related Work

2.1 Most Common Sense

Starting with the first method, Most Common Sense finds the sense of a word in the context by simply going through all the synsets for that word and then selecting the most common synset (which is the first synset in the WordNet). The method does not consider the context and is the most basic of all methods used for the WSD. In our implementation of this method, we start by giving one sentence at a time, and for each word in the sentence, we take the most common synset for that word from the WordNet which is the first synset from the list of synsets corresponding to each word. After each word has been disambiguated, we match it with the ground truth, which is the actual synset for that word in the context, and report accuracy for the datasets [9].

2.2 Lesk Algorithm

The main idea for the Distributional Lesk comes from the Lesk algorithm, which is another WSD method based on knowledge-based learning, that finds the sense of a word in a context by considering the maximum overlap between each of its senses and the words in the context. We implemented this method in the baselines to later compare it with the method described in the paper. For each ambiguous word in a sentence, we start by taking all the synsets that correspond to this word in the WordNet along with the context which is the rest of the words in the sentence excluding this word. For each synset, we find its definition from the WordNet, and then the intersection is taken between the set of all words in the definition of the word and the set of words in the context. The number of words in the intersection gives a score for that synset and this process is repeated for all synsets. Finally, the one with the highest score is chosen to be the sense for the ambiguous word according to our model. To report accuracy, we maintain a variable that counts each of the correctly predicted senses to the total number of ambiguous words in the dataset [10]. The method is tested upon three datasets containing words from the English language and the results are reported in Section 6.

2.3 Word Embeddings

Initially, words were represented via one-hot encoded vectors to be used for machine learning algorithms, but they do not carry any semantic meaning that can be associated with the word. In the past few years, word embeddings have made much progress in representing words as contextual feature vectors for large unlabelled corpora which can be used to find similarities among words and capture lexical relationships [11], [12]. As each word has exactly one-word embedding corresponding to it, the fact, that one word can have different senses according to the context in which it is used, is neglected.

3 Methodology

The goal of our WSD algorithm is to find the sense of each polysemous word in a sentence. For this, we take each sentence as input having words $w_1, w_2, \ldots, w_i$, where each of these words w_i has a set of senses associated with them. For each sense s of each word w, we first find the similarity of the lexeme (combination of sense s with word w) [13] with the context and add it to the similarity of the gloss with the context. In our implementation of this algorithm, we started with passing each sentence as input to our model and the words of the sentence are stored in a set to avoid repetition of words. Since we are only interested in polysemous words, we only perform these steps on such words. Each ambiguous word in the sentence becomes our target word for finding sense, and the rest of the words in the sentence are context words. Since the ambiguous word has more than one synset associated with it, we extract all these synsets from WordNet (here referred to as sense).

We take each sense of the word and extract the lexeme embedding $L_{s,w}$ and gloss embedding G_s associated with it. To get lexeme embeddings $L_{s,w}$, we used AutoExtend [14] to store embeddings of lexemes in a dictionary having a vector size of 300. Since not all senses of all words will have their lexeme embeddings available, we take all possible lemmas for the sense s of word w, and the one which has embedding available, we store it as $L_{s,w}$. Gloss embeddings G_s are calculated for each sense by taking embeddings of all the words in the definition and then returning the average of these

embeddings. Similarly, we find the Context embeddings C_w by taking the average of the embeddings of all the words in the context.

After getting the above defined three embeddings, namely $L_{s,w}$, C_w and G_s for each sense s of word w, we find the cosine similarity between the gloss vector G_s and the context vector C_w and between the lexeme vector $L_{s,w}$ and the context vector C_w. The Cosine similarity helps us in finding words that are like others. Finally, each sense s of word w gives a score based on which we select the meaning of the word. The sense with the highest score is chosen and is calculated as given in equation (1) below:

$$Score(s, w) = \cos(G_s, C_w) + \cos(L_{s,w}, C_w) \tag{1}$$

There may be some cases where either of these embeddings is not available, in that case, we just calculate one part of the sum from the equation (1). Another important practice while performing WSD is that we first sort the words in the sentence according to the number of senses each word has as it is easier to disambiguate words with lesser senses first [15]. When we are trying to disambiguate one word, the words in the context may be ambiguous themselves and if we disambiguate words with fewer senses first then it becomes easier to disambiguate words following them. For this purpose, we have maintained a dictionary in our implementation that stores the sense predicted for the words already disambiguated and is used, while we are dealing with other ambiguous words, for finding the context embeddings.As per [16], for a sense s of a word, its gloss is further expanded by taking glosses of all the hyponyms of the sense s which are used to get gloss embedding G_s.

4 Datasets

We have worked on the three datasets in order to measure the performance of our algorithm as well as for comparison with the baseline methods. These datasets are SemCor3.0, Senseval2, and Senseval3, each of them containing English sentences. All three datasets store the sentences in a similar way so we will go through how we extracted data from one of these datasets. The dataset contains tag files and each of these files have data stored in it in the form of tags. We used BeautifulSoup to read those tags and extract useful information such as word, lemma, POS tag, wordnet synset number, etc from them in order to get both sentences for experiment upon as well as ground truth for measuring accuracy. The POS tagging for the dataset did not match the one used by WordNet and hence we wrote a mapping function that converts these pos tags. Finally, a random selection of 5000 sentences was done on the SemCor3.0 dataset whereas all the sentences were taken from the other two datasets. A list consisting of these sentences was initialized. For the algorithms, we need this list directly as well as we wrote another function that simply fetches one sentence from one tag at a time. The list consisting of tags is used for getting ground truth for each word whereas the list consisting of words corresponding to each sentence is used for disambiguating words. The sense inventory used for our datasets is Princeton WordNet, where almost all words have their senses available.

5 Experiments

The Performance of our algorithm was tested on the three datasets mentioned in the Section 4. For each dataset, we have randomly selected a certain number of sentences (5000 for the SemCor and almost all sentences from the other two datasets). The sentences are passed one at a time and all the words are disambiguated in that sentence which is ambiguous before we move to the next sentence. For each ambiguous word, we find three embeddings corresponding to each sense of the word namely lexeme embedding (using AutoExtend), Gloss embedding and Context embedding as mentioned in Section. The similarity measure used in our algorithm gives us a score that tells us which sense a particular word carries in that context. After we loop through all the sentences in that dataset, we calculate how many of the total ambiguous words we predicted the correct sense and report the accuracy for each of the datasets.

In our implementation of the algorithm, we worked only with the English sentences in an all-words setup. The word embeddings used are from Word2Vec which are pre-trained over the Google news dataset, which contains about 100 billion words. The word embeddings so obtained is a 300-dimensional vector. We compare the results with the two baselines mentioned in Section 2, namely Most common sense and Plain Lesk. Even though the idea behind the Distributional Lesk comes

from Plain Lesk, we see how the introduction of word embeddings can significantly improve results. The next section gives a brief description of the results obtained from our experiments.

6 Results

The results of the evaluation of our algorithm as well as the two baselines Most Common Sense and Plain Lesk are presented in Table 1. All the methods are evaluated on three datasets SemCor, Senseval2, and Senseval3 for English sentences. The best performance is based on the accuracy of the algorithms on these datasets. Accuracy is calculated by dividing the total number of words that have been disambiguated accurately by the total number of polysemous words.

Table 1: Results of the algorithms on different datasets

	SemCor3.0	Senseval-2	Senseval-3
Most Common Sense	37.8%	37.26%	36.9%
Plain Lesk	32.05%	30.68%	29.86%
Distributional Lesk	46.42%	42.69%	40.21%

As can be seen in Table 1, Distributional Lesk gives state-of-the-art performance on all the three datasets with the best on SemCor3 dataset with an accuracy of around 46%. Among the two Baseline methods, Most common sense performs better than the Plain Lesk as shown in Table 1.

7 Extensions

We have seen the results in the previous section of the Distributional Lesk. Though it has performed better than the two baselines: Most Common Sense and Plain Lesk, we further extend our experimentation by adding extra conditions and checking how this affects the performance of our WSD method. We start by removing stop words and punctuations, followed by training the pre-trained word embeddings on our datasets, further we experimented with the hyperparameter tuning for these pre-trained word embeddings. The word embeddings used so far are word-based and hence we then also experimented with character-based and sentence-based embeddings such as Flair embeddings and SBERT. The effects of these extensions have been reported in the following sections.

7.1 Effects of Removing Stop words and Punctuations

We first started by removing stop words and punctuations from both the gloss and the context. This is done by simply removing the punctuations and stop words from the gloss of the sense we are working with, which is the definition corresponding to a particular sense of the ambiguous word, and from the set of context words. In the set of context words, stop words might not help in understanding the similarity with the ambiguous word and are therefore unnecessarily used when finding the context embeddings. A similar argument can be given while finding the gloss embedding. In our implementation of the algorithm, we passed an argument to the function which when False does not remove stop words, and when True it removes stop words. The effect of this on all the datasets has been recorded but contrary to our assumption, it does not make much difference in the accuracy. The accuracy for both cases (with or without stop words) is roughly the same. The results are shown in Table 2.

7.2 Evaluating results on categories of text in Brown corpus

The SemCor dataset consists of texts from many categories like News, entertainment, books etc. In this section, we will be testing the performaance of the Distributional Lesk on each of these categories.

As can be seen in Figure 1, the accuracy on all the categories lies between 25% to 40% which is roughly the same as we get on the entire dataset.

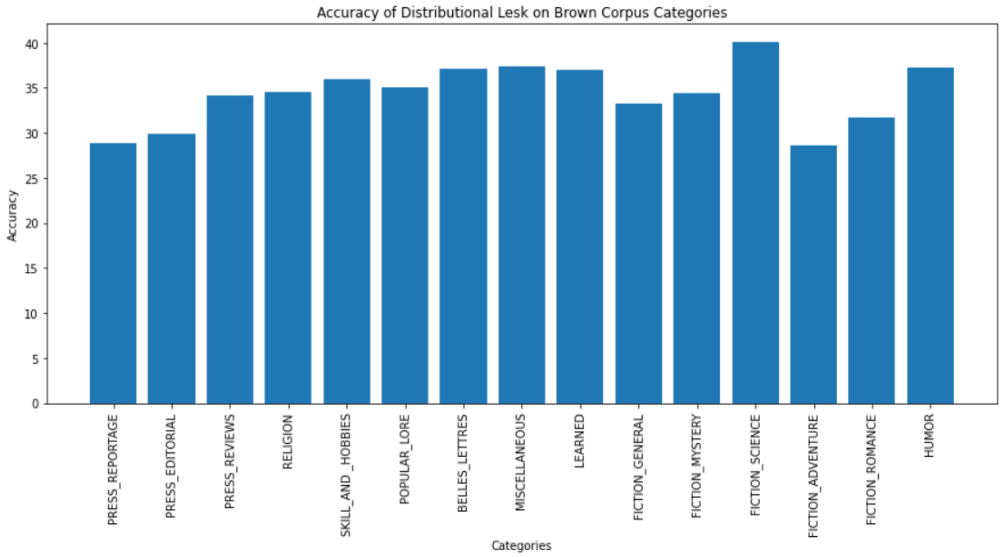

Figure 1: Results of distributional Lesk on Brown corpus categories

7.3 Effects of training embeddings on our datasets

The word embeddings we have used so far were pre-trained on the Google News dataset and there is a possibility that some words in our datasets might not have their embeddings available. Also, these embeddings when trained, have different features associated with each dimension of the embedding vector and as these features help us in finding similarities among words which then are used to disambiguate words, we have trained these pre-trained embeddings on the dataset consisting of sentences combined from all three datasets. As shown in Table 2, this does not help in improving the accuracy of our model rather the accuracy has dropped. As these were simply pre-trained embeddings, in the next section we hyper tune some of the parameters and see how it affects the accuracy.

7.4 Tuning the parameters in the pre-trained embeddings

As done in the previous section, we have only trained the embeddings on our datasets without making any changes in the parameters of the pre-trained embedding model. Here we will experiment with these parameters and see the effects on varying min_count and window. The parameter min_count ignores all words that have a frequency lower than it whereas the parameter window gives the number of maximum words on the left and right side of the word to be used while training. Upon hyper tuning, the best values for these two parameters found were 2, 8 for the min_count and window respectively. As compared to the original model, the accuracy is still not better but it has performed better, than the pre-selected parameters used in the previous section.

7.5 Using character-based contextual embeddings: Flair embeddings

When using word embeddings, each word has one embedding associated with it even though it might have different meanings with respect to the context. In the character-based embeddings, they are trained without any explicit notion of words and thus fundamentally words are models of sequences of characters, and the embeddings are contextualized by their surroundings, meaning the same word will have different embeddings depending on the context. In particular, we have used stacked embeddings for our experiment where we have combined three embeddings namely Word2Vec embedding, forward and backward flair embeddings. Instead of passing one word at a time to obtain its embedding, we give the whole sentence to our stacked model, which finds embeddings for each word according to the text surrounding it, and then we can fetch individual embeddings for each word. Since here we train to the whole sentence at once to get embeddings of each word, the words that have been disambiguated previously, are replaced with their senses in the set of context words to get the embeddings [17].

7.6 Using sentence-based embeddings: SBERT

Finally, we have used sentence-based embeddings as part of our extensions to see the performance of our algorithm. Here we have a pre-trained model from the Sentence transformers that we will be using. The difference from the previously defined embeddings is that there we get embedding for individual words irrespective of whether we passed a word or sentence to our embedding model but here we get embedding for the entire sentence as one vector. For the gloss embedding, we pass the definition of the sense to the transformer and for the context embedding, we take all the words in the context (which acts like a sentence here even though the words are shuffled and don't make much sense) and pass it to our transformer. The words whose senses have been found according to our algorithm, later on when these words are passed as context words, we replace them with their senses in order to find context embeddings [18].

8 Discussion

The Extensions performed on the Distributional Lesk algorithm in Section 8 have increased the accuracy of the model the most in the case of Flair embeddings and SBERT. As can be seen in Table 2, the removal of stop words does not impact the accuracy much, for the SemCor dataset, accuracy has decreased slightly whereas for the other two datasets it has increased a bit. On tuning the parameters for the pre-trained embedding that is trained on our dataset, we see a slight increase in the accuracy as compared to directly using pre-trained embeddings. Both Flair and SBERT have significantly improved the accuracy and are on top of all the extensions performed.

Table 2: Results of the Extensions on the three datasets

	SemCor3.0	Senseval-2	Senseval-3
Distributional Lesk	46.42%	42.69%	40.21%
Stop words Removal	46.27%	43.88%	40.96%
Word2Vec Trained embeddings	43.13%	36.82%	37.11%
Parameter tuning in pre-trained embedding	44.28%	36.37%	39.27%
Flair Embedding	47.42%	44.69%	41.21%
SBERT	48.72%	45.31%	42.55%

9 Conclusion

In this Report, we have presented an extension of the Lesk algorithm which is a knowledge-based WSD method that computes the similarity of word senses to the context in which it occurs. Even though the basic idea to consider the overlap between the sense of the word and its context is followed in both the algorithms, instead of using the intersection of the gloss of the sense with the context, we have used sense, gloss, and context embeddings. The method proposed in this paper has performed better than both the Plain Lesk as well as another baseline method used for comparison namely Most Common Sense. Our experiments were performed on datasets consisting of sentences from the English language. We further extended our experiment by varying some of the parameters used in the original model to see the effects on the accuracy gained by the model. These extensions were mainly removing stop words, and using trained/pre-trained word embeddings, using character and sentence-based embeddings. The main advantage this model has over supervised learning models is that it does not require a large amount of manually annotated data in order to train the model. It just requires unlabelled text and the definition of the senses of the words. The model is simple, easy to implement, and can be extended to other domains as well.

References

[1] Snyder, B. and M. Palmer (2004, July). The english all-words task. In R. Mihalcea and P. Edmonds (Eds.), Senseval-3: Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text, pp. 41–43.

[2] Pradhan, S. S., E. Loper, D. Dligach, and M. Palmer (2007). SemEval-2007 task 17: English lexical sample, SRL and all words. In Proceedings of the 4th International Workshop on Semantic Evaluations, pp. 87–92.

[3] Navigli, R. and M. Lapata (2007). Graph connectivity measures for unsupervised word sense disambiguation. In Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 1683–1688.

[4] Navigli, R. (2009, February). Word sense disambiguation: A survey. ACM Computing Surveys 41(2), 10:1–10:69.

[5] Zhong, Z. and H. T. Ng (2010). It makes sense: A wide-coverage word sense disambiguation system for free text. In Proceedings of the ACL 2010 System Demonstrations, ACLDemos '10, pp. 78–83.

[6] Escudero, G., L. Marquez, and G. Rigau (2000). An empirical study of the domain dependence of ' supervised word sense disambiguation systems. In Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, pp. 172–180.

[7] Agirre, E., O. L. De Lacalle, and A. Soroa (2009). Knowledge-based WSD on specific domains: Performing better than generic supervised WSD. In Proceedings of the 21st International Joint Conference on Artifical Intelligence, pp. 1501–1506.

[8] Lesk, M. (1986). Automatic Sense Disambiguation Using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone. In Proceedings of the 5th Annual International Conference on Systems Documentation, SIGDOC '86, New York, NY, USA, pp. 24–26. ACM.

[9] https://github.com/JGuymont/lesk-algorithm/blob/deaaa8fe451a376d6664e9283d64cb5863763b7c/lesk/baseline.py

[10] https://en.wikipedia.org/wiki/Lesk$_algorithm$

[11] Turney, P. D. (2006). Similarity of semantic relations. Computational Linguistics, Volume 32, Number 3, September 2006.

[12] Mikolov, T., W. Yih, and G. Zweig (2013). Linguistic regularities in continuous space word representations. In Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, pp. 746–751.

[13] Rothe, S. and H. Schutze (2015). AutoExtend: Extending word embeddings to embeddings for synsets ¨ and lexemes. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pp. 1793–1803.

[14] Rothe, S. and H. Schutze (2015). AutoExtend: Extending word embeddings to embeddings for synsets ¨ and lexemes. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pp. 1793–1803.

[15] Chen, X., Z. Liu, and M. Sun (2014). A unified model for word sense representation and disambiguation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, pp. 1025–1035.

[16] Banerjee, S. and T. Pedersen (2002). An adapted Lesk algorithm for word sense disambiguation using WordNet. In Proceedings of the Third International Conference on Computational Linguistics and Intelligent Text Processing, pp. 136–145.

[17] https://github.com/flairNLP/flair/blob/master/resources/docs/TUTORIAL$_3WORD_EMBEDDING.md$

[18] https://www.sbert.net/