COMP550 - Natural Language Processing - PA 2

Name: Ada Tur , **McGill ID**: 261069241

November 22, 2023

0.1 Introduction

This report details a series of experiments and models for a word sense disambiguation task. Word sense disambiguation is the challenge of, given a lemma and a context sentence, identifying a correct sense of the word, such as river bank vs. money bank. Four approaches were conducted and evaluated: a baseline approach that automatically assigns the most frequent sense for the lemma, Lesk's algorithm, which analyzes contextual data for the lemma and compares with dictionary definitions, a bootstrapping method similar to Yarowsky's algorithm that uses an iterative process with a Multinomial Naive Bayes model, and an approach using word embeddings and cosine similarities.

0.2 Baseline

The baseline method, which simply takes the most frequent sense, performed particularly well in this experimentation. On the evaluation set, an accuracy of 49.5% was reached, and on the test set, 50.6%.

0.3 Lesk's Algorithm

Lesk's algorithm is another model evaluated for word sense disambiguation. Input sentences are lemmatized, and stop words are removed, and run through Lesk's algorithm from the Natural Language ToolKit (NLTK). The algorithm works by calculating the level of overlap between the dictionary definitions of words, and analyzing the co-occurring words around a target word to be disambiguated, and choosing a synset accordingly. The performance of Lesk's algorithm was worse than the baseline, though consistent with the observations of others; 29.9% on the validation set and 29.3% on the test set. A possible reason for the decrease in performance is, firstly, that the most frequent sense is most often the correct sense, and, second, that the analysis of the words surrounding a target word can confuse the model - some of the senses are very similar to each other, and it can be difficult for the model to be confident in one particular sense over others, and to be entirely accurate. Nevertheless, Lesk's algorithm offers insight into a method of disambiguating word senses using lexical information.

0.4 Bootstrapping Method

The bootstrapping method utilizes a classifier that trains on a large corpus, and a small seed set, and disambiguates word senses accordingly. Firstly, the algorithm extracts example sentences from an available set from WordNet, though the number of examples is typically only a few. These examples are labelled, which allows for easy usage in a seed set, though more examples could be added for better performance. The Brown corpus is utilized as a source of unlabelled language data for the model to additionally train on. A list of words to be disambiguated are selected based on frequent words and their respective senses from the validation set. Then, for 3 iterations, a Multinomial Naive Bayes classifier trains on the examples and the Brown corpus and makes predictions on one word to be disambiguated accordingly. At each iteration, the sentences where the model had the highest confidence in the correct word sense are added to the training set, and the process is repeated. For each word, it was important to see some level of improvement in accuracy. The words selected were chosen from the available senses, where the words with improvement over the iterations were reported. The words were: "claim", "degree", "state", "time", "dinner", and "end". Below is a table reporting the improvements over each word for 3 iterations, where one example per sense with a confidence of over 60% is added to the training set:

	Iteration 1	**Iteration 2**	**Iteration 3**
claim	0.1	0.3	0.3
degree	0.416	0.833	1.0
state	0.75	0.75	0.792
time	0.333	0.167	0.333
dinner	0.4	0.2	0.4
end	0	0.143	0.286

Overall, there is some improvement in the words, though most other words saw accuracies remaining the same. The reason for such great differences between some accuracies (such as 0.25 to 0.5) was that each sense had very few instances in the testing set, so accurately classifying one more lemma could have increased accuracy by over 20%. It should be noted that changing the number of iterations and the threshold gave varying results. For instance, "claim" saw an increase in accuracy from 0.10 to 0.80 with a threshold of 60% and 5 iterations, "degree" saw an increase in accuracy from 0.42 to 1.00, with the parameters in the table above, "state" was improved from 0.75 to 0.79 with the parameters in the table above, "time" saw an increase from 0.33 to 0.50, with a threshold of 70% and 5 iterations, "dinner" from 0.4 to 0.6 with a threshold of 70% and 4 iterations, and "end" from 0.0 to 0.29 with a threshold of 60% and either 3, 4, or 5 iterations. An example of a sentence added to the train set is for "state", the following line was added: "the state public work department is accused of having spent $8,555 to build a private beach for a state judge on his waterfront property". Many words had accuracies remaining the same mainly because MultinomialNB is a Bayesian algorithm, and we provide roughly the same number of examples for each sense, and there are few examples in our seed set, so the model ends up performing weakly. In order to alleviate this, we only considered the first 4 senses of each word, as we assume that they are the most frequent senses. Overall, the bootstrapping method showed improvement in disambiguating some words, though, with a larger seed set, it would have been much more powerful. Had more examples been added for each sense, and a greater difference between each sense, the bootstrapping method may have seen more increases in accuracy.

0.5 Word Embeddings + Cosine Similarity Method

For the final method, a Word2Vec and cosine similarity approach is used. The basis for this method was converting sentences to word embeddings using the Word2Vec model from gensim with glove-twitter-25, a set of pre-trained vectors using a large corpus of tweets, and conducting cosine similarity to extract the most similar sense (Note for the grader: you may need to run the following command in order to be able to download the Word2Vec model: "bash /Applications/Python*/Install Certificates.command"). Embeddings prove to be very powerful tools; we are able to extract semantic meaning behind each word from a very large corpus of data, and represent this meaning numerically. Better embedding models could have been used, in which cases different results may have been presented. For each sense of an instance token, we computed an embedding by taking the average of the token embeddings of its definition sentence (we call this the sense embedding). Then, for each specific instance, we computed an embedding by taking the average of the embeddings of tokens in context sentence (we call this the instance embedding). We then computed the cosine similarity between the instance embedding and all sense embeddings, and assigned the sense whose embedding is the closest to the instance embedding to the instance. Experimentation was also done considering the limitation of possible senses; many senses of a lemma weren't present in the validation or test set. There are many, many rare senses of a word, but when we consider all of them, we add lots of unnecessary options for the model, as it cannot tell which sense is more frequent, or properly distinguish them enough. When we limit to expected senses, we give the model a more realistic set of options to classify from. Below is a table comparing the difference between limiting senses to those present in the validation set and no limitation, and the resulting performances, where the y-axis labels represent if the senses were limited by the dev set or none, and x-axis labels represent which set was run through the model with the limitation.

	Dev Set	**Test Set**
Dev	0.706	0.707
None	0.284	0.293

An interesting observation is that the validation set and test set don't share a large amount of senses. Knowing the possible senses greatly helps, but this wouldn't be possible with unlabelled data, and is therefore not an honest way of conducting this experimentation. Without limiting the senses, we have performance similar to Lesk's algorithm, though not quite as good. A potential reason for why the performance in this case wasn't as high as hoped is likely due to the fact that for most word senses, there is a large level of similarity between their definitions. Though the occurrences of these two senses may be different, there is considerable overlap between the definitions and examples of the two. As a result, cosine similarity can often get very confused between two similar senses.

0.6 Discussion and Conclusion

Overall, the four approaches represented various ways of completing word sense disambiguation and analyzing language data. A large challenge for many of the approaches was the presence of very similar senses that were represented as separate entities. In an ideal world, all words to be disambiguated would have very different senses, such as a river bank vs. a money bank. However, many of the senses for a word were very similar. For instance, two senses of "game" have definitions "animal hunted for food or sport" and "the flesh of wild animals that is used for food". The word embedding approach performs the best out of our experiments, and comparing definitions of words, examples, semantic representations, and other language data proves to offer many insights to how language can be very difficult to disambiguate, and how we can most effectively complete this task.