Programming Assignment 2: Word Sense Disambiguation

Hamza Rashid
** See baselines.py for instructions **

November 16, 2024

1 Introduction

Word Sense Disambiguation (WSD) is the task of determining the correct sense of a word in context. In this assignment, `Most Frequent Sense (MFS)` and `Lesk's algorithm` are tested as baselines, and we present two additional methods using pre-trained language models: `Named Entity Overlap (NEO)` and `Transformer embeddings L2 distances (TeL2d)`. As per assignment instructions, we used the SemEval 2013 Shared Task #12 dataset, using WordNet v3.0 as the lexical resource. This report presents the methodology, results, and analysis of these experiments. Due to the small size of our dataset and its specialized content, the following results will not necessarily hold in a general setting. We used the dataset split provided in the starter code for development and testing. The baselines were implemented with NLTK's tokenizer, stopword corpus, Lesk method, and WordNet synsets. These resources (except for the Lesk implementation) are also used in the custom methods for preprocessing and model evaluation (looking up synsets). In addition, both of the custom methods were implemented with a Python NLP library called `flair`.

2 Methods

2.1 Baseline Methods

For MFS, we utilized NLTK's `wordnet.synsets()` method, choosing the synset whose sum of lemma frequencies was the largest. This method yielded 55.93% accuracy on the test set that was provided. Preprocessing considerations included lowercase conversion and underscore removal when dealing with multi-word phrases, the former decreasing the accuracy and the latter being necessary for all methods in this assignment. Keeping with the baseline treatment, we did not supplement this method with part-of-speech tagging. Moving on to Lesk, its use of token-level comparisons led to various preprocessing experiments on the dev set: stopword removal, punctuation removal, and lowercase conversion. Since the dataset was biased toward politics and economics, we saw a performance decrease with lowercase conversion. Punctuation and stopword removal improved the accuracy, trading quantity for more meaningful overlap. In the end, Lesk's best accuracy of 30.41% was achieved with stopword and punctuation removal, and case retention. The relatively poor performance is likely caused by the specialized dataset.

2.2 Custom Method 1: NEO

NEO is a variant of Lesk, computing overlap between the named entities of both the context and the synset example text or definition (when no example exists). Our hypothesis is that Lesk misses meaningful overlap due to the presence of synonyms (of the context words) in the synset text, and that the use of named entities can resolve this oversight. Our intuition is that an example is likelier to contain meaningful entities to be compared against, which proved to be effective on the dev set. Notable challenges in this method include sparsity of entities in both the synset texts (resorting to the definition when an example was not available) and the sequence-labeling models' tag sets. As a result, we experimented with various models varying in size, speed, and number of classes. The pre-trained models were based on the BiLSTM-CRF (IOB) sequence-labeling architecture proposed in *"Contextual String Embeddings for Sequence Labeling"*, by Akbik, Blythe, and Vollgraf (2018), namely: `Flair English 4-class NER` and `Flair English 18-class NER`. We evaluated the `base` and `fast` variants of the former and the `large` and `fast` variants of the latter on the dev set. We got the best results with the 18-class fast model, scoring 63.40%. This likely owes to the model reducing sparsity but not being too accurate and causing unwanted overlap. The

4-class model predicts tags `PER, LOC, ORG, and MISC`—person, location, organization, and other, respectively. In addition to the 4-class tags, the 18-class model predicts some tags that are relevant to our task, such as `GPE` (geo-political entities) and `NORP` (nationalities or religious or political groups). This reduces sparsity in the dataset's tags, making it easier to resolve the sense of a domain-specific phrase. On the other hand, general-purpose words that are collocated with domain-specific phrases are often harder to resolve since their definition and example texts will rarely use political terms.

2.3 Custom Method 2: TeL2d

For this method, we use a pre-trained transformer model to generate document embeddings and select the synset whose example text embedding minimizes L2 distance with the context's embedding. As this method is less intuitive, we provide sample output (of Python pseudocode) below:

```
# disambiguate golden_retriever
context = flair.Sentence("a distinguished gentleman and golden_retriever named Marshall")
definition = flair.Sentence("an English breed having a long silky golden coat")
bert_embedder.embed(context), bert_embedder.embed(definition)
L2_dist = numpy.linalg.norm(context.embedding - definition.embedding)
print(L2_dist) # want to find the example/definition minimizing this
>>> 5.690494
```

Our hypothesis is that a transformer's contextualized embeddings capture sufficient semantic information to map a sentence close to the gold synset's text for each of its words, at least in a subspace of the model's representational space. This method continues in the spirit of NEO, attempting to generalize Lesk to account for implicit similarities. Since we are working with large pre-trained models, we expect them to be able to implicitly encode entities and other latent information that could be useful for our task. In addition to the preprocessing considerations from the previous methods, we did model selection and hyperparameter tuning on the dev set. Punctuation and stopword removal had an adverse effect on accuracy, meaning that higher quality embeddings required textual richness. Thus, the underlying dataset's lemmatization was not ideal, but for the purposes of the assignment, we left it unchanged. On the dev set, we evaluated the accuracy of each layer of the following models: `bert-base-cased (12 layers), all-mpnet-base-v2 (12), and all-MiniLM-L6-v2 (6)`. Treating BERT as a baseline, we found that the cased version outperformed the uncased one, which is consistent with the preprocessing experimentation. `All-mpnet-base-v2` and `all-MiniLM-L6-v2` are both fine-tuned on a 1B sentence pairs dataset, using contrastive learning objective: given a sentence from the pair, the model should predict which out of a set of randomly sampled other sentences, was actually paired with it. As this pre-training task is adjacent to ours, these models seemed promising. On the dev set, the most accurate model was `all-MiniLM-L6-v2`, with its 6^{th} layer scoring 51.03%, and falling shortly behind the approach where we take the mean of all layers to compute the L2 distance, scoring 51.54%. One might suspect that the larger, cased models, while encoding more information, had more noise.

3 Results and Conclusion

On the test set, MFS, Lesk, NEO (`18-class fast model`), and TeL2D (`all-MiniLM-L6-v2`) scored 55.93%, 30.41%, 57.86%, and 46.83%, respectively. While NEO performed the best, it is important to note that the First Sense Baseline (select the first sense returned by NLTK's `synsets()` method) scored 61.93% on the test set. Thus, one might postulate that the sparsity of entities accounted for the higher accuracy. We attribute the low performance of TeL2D to the specialized dataset, causing the same issues that were discussed in the analysis of NEO on the dev set. Namely, the transformer embeddings for the dataset's contents favor domain-specific words, while the representation for a general-purpose word's definition text is likely far away from its context's embedding. One can expand upon the TeL2D method by training the transformer model on the corpus. As for NEO, fine-tuning a sequence tagger to predict more classes can be helpful for treating the entity sparsity, and one can take this even further by using an IOBES tagging scheme. However, it is more important to work with a dataset that is larger and broader (in domain and subject) than the one used in this assignment. The results in this assignment cannot be used to infer how these models would perform in a general setting. Overall, WSD is a task that combines hand-crafted resources with intelligent systems, and it is important to consider how the relationship between the dataset in question and the gold label resource impacts the method used for making predictions.

References

- Navigli, R., & Jurgens, D. (2013). SemEval-2013 Task 12: Multilingual Word Sense Disambiguation. *Proceedings of the 7th International Workshop on Semantic Evaluation.*
- Akbik, A., Blythe, D., & Vollgraf, R. (2018). Contextual String Embeddings for Sequence Labeling. *Proceedings of the 27th International Conference on Computational Linguistics*, 1638–1649.
- WordNet Documentation: `https://wordnet.princeton.edu/documentation/senseidx5wn`
- Hugging Face: Sentence-Transformers Model `all-mpnet-base-v2`. Retrieved from `https://huggingface.co/sentence-transformers/all-mpnet-base-v2`.
- Hugging Face: Sentence-Transformers Model `all-MiniLM-L6-v2`. Retrieved from `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`.
- Flair NLP: `https://flairnlp.github.io/docs/intro`.