SAPIENZA
UNIVERSITÀ DI ROMA

Sense Embeddings

NLP HOMEWORK 2 REPORT

MAKINWA, Sayo Michael
[1858908]

04.06.2019

1.0 Introduction

The task is to train sense embeddings using annotated English sentences extracted from a given corpora and a given set of sense mappings, and implementing a word-similarity task to test the model. The image below is a graphical representation of the full pipeline of my implementation:

Fig. 1. Graphical description of the full pipeline

2.0 The Dataset and Data Preprocessing

Data is more important than the underlying neural architecture used to learn the embeddings (Levy et al. 2017, Levy et al. 2015), for this reason, I spent a considerable amount of time cleaning up the data as best as I could, even at the point of parameter tuning!

For this implementation, I used both the high precision and high coverage versions of the EuroSense corpus, comparing their individual and merged performances. I parsed each of these versions of the corpus to extract English sentences from them, and for each of these sentences, the English annotations given were used to replace anchor words with the corresponding **lemma_BabelNetID**.

The data is pretty noisy, and while trying to improve the model, the changes I made to how the data was parsed – whether or not all texts were converted to lower case, some errors in replacing annotations, whether or not wrong annotations were filtered out – had effects on the score (all of which I am unable to record here); interestingly, the correlation score was better when an error led to wrong replacements of annotated words for some hyper-parameters, but performed worse on tuning. The following are the important things to note in how I parsed the corpus (everything described in the following parts basically explains *corpora.py*):

1. The **cases of annotations overlap** in the corpora are generally in three categories:
   a. **Direct duplicates** – lemma and synset IDs all the same. For this, I implemented lists for each sentence that is populated with encountered annotations that had not been previously added to the list.
   b. **Duplicates with different (and wrong) synset IDs**. For this, I extended the condition for populating the lists in (a) above such that, the correct lemmas for each annotation are fetched using the bn2wn_mapping.txt and the WordNet NLTK API, and the new annotation is added only if it's lemma matches one of the lemmas the synset ID fetches. In addition, the replacements were done for full words. This ensures that whatever annotation is taken for the word, it will be a correct one.
   c. **Overlaps with varying length of anchor words**. As written in the README.txt file, and also using the lists described in (a) above, for each new annotation, I first verified that there is any overlap at all between any of the words in the annotation and any of the annotations already accepted in the list, then for the high precision data, I checked which has a longer anchor text – if the new annotation has the longer anchor text, it **replaces (not appended to)** the annotation it overlaps in the list. The same goes for high coverage data, except that the condition for acceptance here is a higher *coherenceScore*.
2. Multiple words lemmas of annotations that are separated with spaces are replaced with underscores
3. All texts were converted to lower cases
4. The hyphens in hyphenated multiple words lemmas are left as they are, as opposed to replacing with spaces as in (2) above.
5. At the later stages of my parameter tuning, I removed stopwords (using the NLTK library) and punctuations.

Furthermore, I noticed that **the corpus is biased** – it is rich in only in senses often used in the parliament and does not contain some senses that are common in contemporary use for some words, and does not even contain any senses at all for some words.

For this implementation, I used both the high precision and high coverage versions of the EuroSense corpora, comparing their individual and merged performances

3.0 Model Description and Parameter Tuning

The sense embeddings (and all the experiments I carried out) were trained using **GenSim Word2Vec**, with the **CBOW** model, trained using **negative sampling**. I started out tuning only five hyper-parameters – window, word vector size, starting alpha, negative, and number of epochs – plus the datatset (to make six altogether) and I generally left the other parameters as the **GenSim Word2Vec defaults**. The changes that resulted from tuning each of the hyperparameters were not exactly linear; some hyperparameters gave better results on their own, but resulted in terribly results when other parameters were tuned. I have recorded a **very few** (that I am able to record) of **some of the results of tuning some of the parameters in Table 1** on the last page of this report.

4.0 Sense Extraction and Word Similarity Task

To **extract the senses**, I walked through all the embeddings the model learned and extracted out the embeddings that belong to word senses, then those embeddings are saved to an *embeddings.vec* file in the required format (see *save_embeddings()* function in *my_utils.py*).

For the **word similarity task**, the two important things I explored were (a.) choosing words to regard as "similar"; and (b.) handling out-of-vocabulary words.

4.1 Similar words

It was explained in section 3.1 of Iacobacci et al how they tried to cover as many senses as can be associated with the words as possible, so I tried quite a number of ways to determine which senses are similar and which ones are not, and for each try, I got a different score for the same model. It seemed as though, the more expansive I made the similar words, the lesser the score I get. Two notable of the many attempts are;

1. For each word, I got a list of the synonyms using WordNet API, then I got all the sense embeddings the model that belongs to any of the word and its synonyms. This runs quite fast and seems to provide a reasonable coverage, so I ended up using this method.
2. For each word, I got every synonyms for the word using WordNet API, then for every sense embedding that matches any of the words in the set of synonyms, I used GenSim's *most_similar()* function to get similar senses for that and add the list to the list of associated senses. So, this method basically gets associated senses for every synonym. Needless to say it took an unreasonably long time to execute, and returned really low scores. I also took note that the *most_similar()* function returned related but not similar words, e.g. **senses for "disagree" were returned as similar to senses for "agree"**.

4.2 Handling Out-Of-Vocabulary (OOV) words

The default score for OOV words is *-1*, but I also tried out assigning zero (0) and 1 to OOV words. Scores returned for default values 0 and -1 were more or less the same, but the scores returned for the default value of 1 was much higher (at times double) the score value for the other approaches. For the results recorded below, I used a default of zero (0).

5.0 A bit of something extra; multilingualism

Given that the one of the end goals (if not the ultimate) of training sense embeddings is to be able to move readily across languages. To this end, I decided extract an Italian dataset from the corpus – sentences and annotations all in Italian – train a model with it, and perform the Word Similarity task on the English given English dataset. To be able to test this idea, I had to get the similar words in a different way since I could not simply check for senses associated with a word in the sense embeddings that are in Italian. For this, I took advantage of the BabelNet ID's that is attached to each of the sense embedding, mapped it to its corresponding WordNet ID, then used that to query for a set of synonyms against which I checked each of the words in the test data (since both sets are in English). I discovered that the Italian dataset requires a bit more cleanup with accented characters, which I did not have enough time to do, so this obviously impacted the results.

6.0 References

Ignacio Iacobacci, Mohammad Taher Pilehvar, and Roberto Navigli. 2015. SENSEMBED: Learning Sense Embeddings for Word and Relational Similarity. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 95–105, Beijing, China, July 26-31, 2015.

Levy Omer, Goldberg Yoav, and Dagan Ido. 2015. Improving Distributional Similarity with Lessons Learned from Word Embeddings. *Transactions of the Association for Computational Linguistics*, vol. 3, pp. 211–225, 2015. Action Editor: Patrick Pantel.

Levy Omer, Søgaard Anders, and Goldberg Yoav. 2017. A Strong Baseline for Learning Cross-Lingual Word Embeddings from Sentence Alignments

https://radimrehurek.com/gensim/models/word2vec.html

https://github.com/sismetanin/word2vec-tsne

7.0 Appendix

		Extracted Sentences	**Full Model Vocab Size**	**No of Senses in Vocab**	Win: 6 **Size: 400** Alph: 0.09 Neg.: 20 Epochs: 10	Win: 6 **Size: 100** Alph: 0.09 Neg.: 20 Epochs: 10	Win: 6 Size: 100 Alph: 0.09 **Neg.: 5** Epochs: 10	**Win: 3** Size: 100 Alph: 0.09 **Neg.: 20** **Epochs: 5**	Win: 3 Size: 100 **Alph: 0.06** Neg.: 20 Epochs: 5
EuroSense English	Merged	3,806,297 (NOT unique)	86,312	37,875	-0.128	0.252	0.255	0.221	0.186
	Precision	1,903,116	87,906	36,911		0.188	0.172		
	Coverage	1,903,181	88,788	58,534		0.207	0.215		
EuroSense Italian	Merged	3,121,096 (NOT unique)	98,255	34,967		0.174			
	Precision	1,560,539	100,459	35,005		0.174			
	Coverage	1,560,557	89,170	51,234		0.174			

Table 1. Table showing some parameters and results

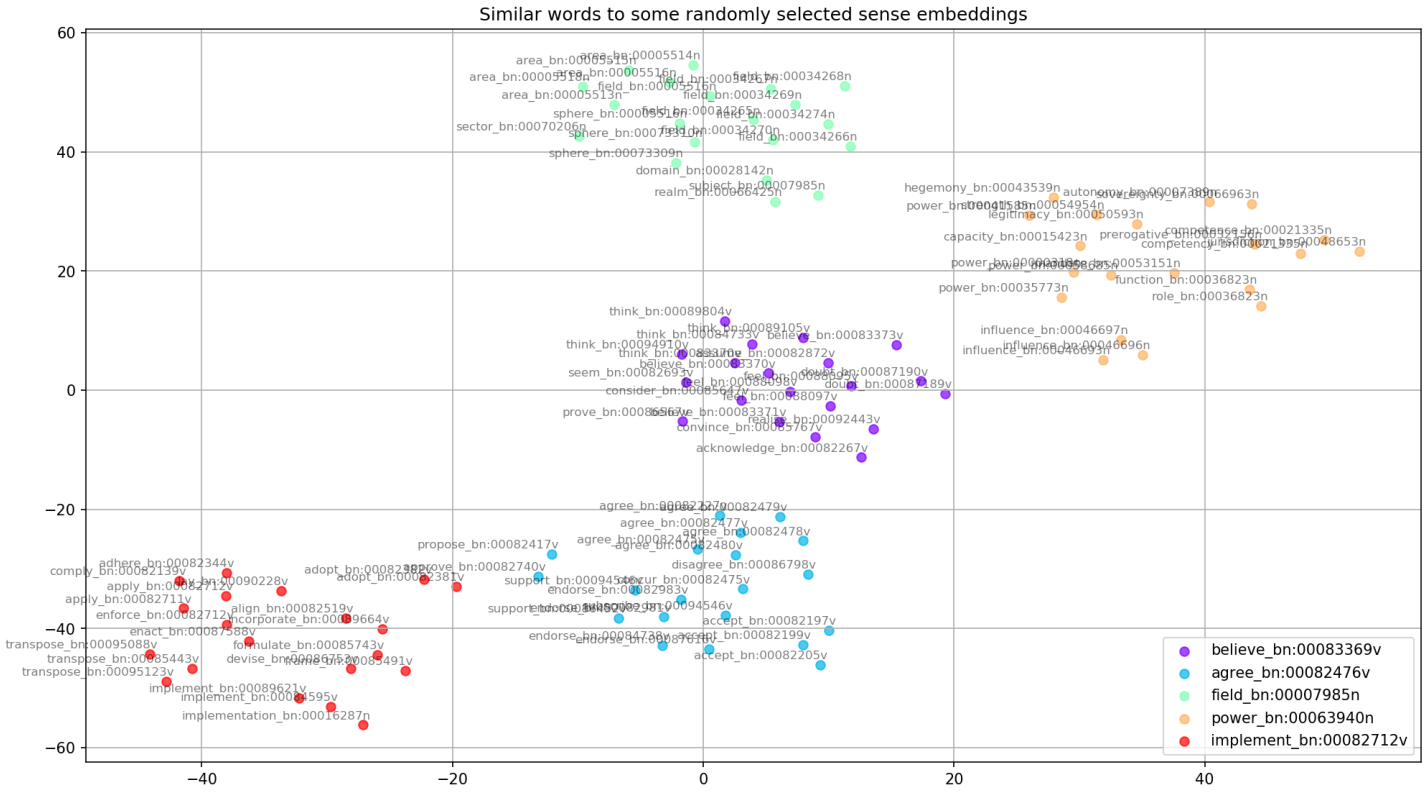

Fig. 2. TSNE plot of similar words to some randomly selected sense embeddings

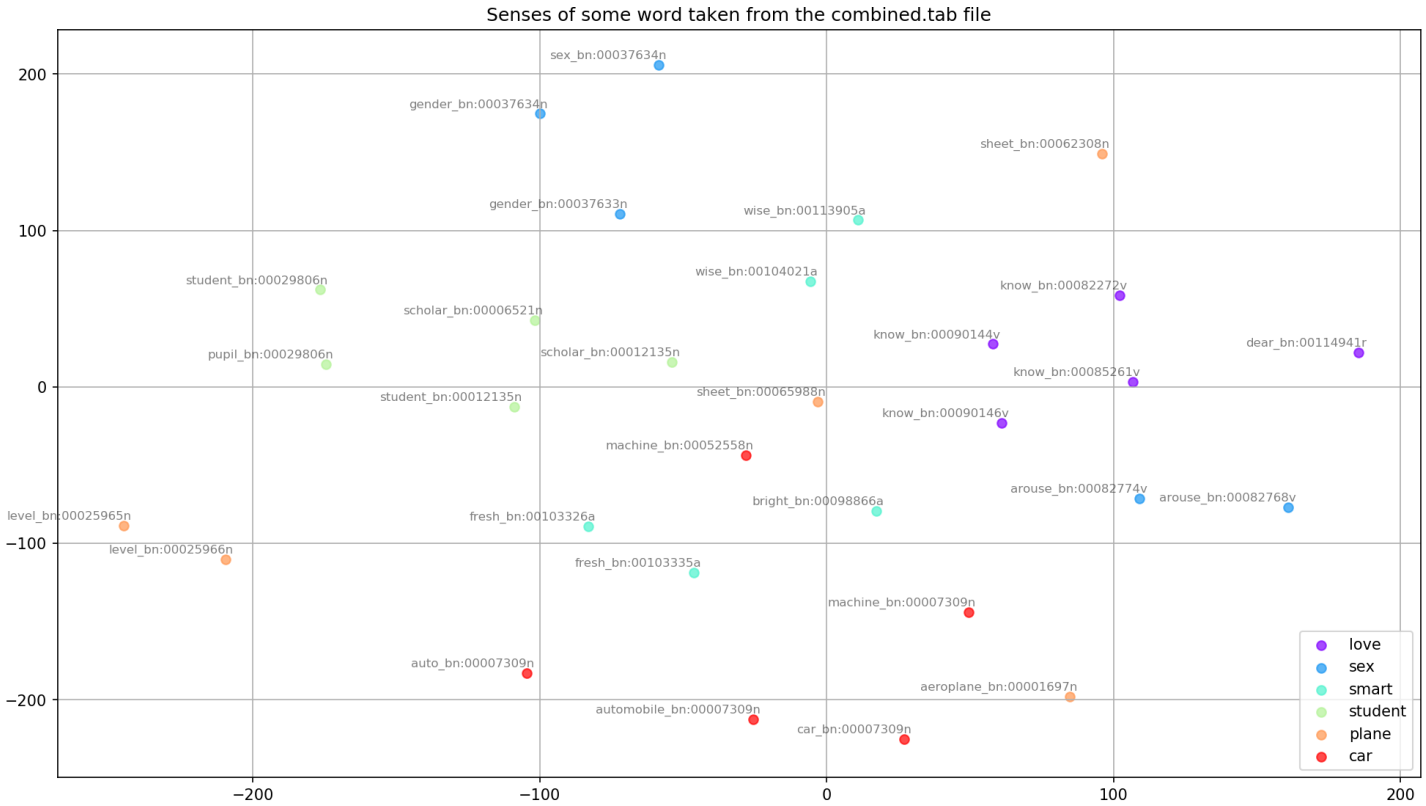

Fig. 3. TSNE plot of some similar senses to some words taken from combined.tab file