SAPIENZA
UNIVERSITÀ DI ROMA

Homework 2 Natural Language Processing 2018/2019

Sense Embedding Model

June 5th

Professor: Roberto Navigli
Student: Anxhelo Diko 1867167
Course: Computer Science LM-18

Task

This task consist in creating our own Sense Embeding, a Word2Vec model or some similar approach to learn how to represent words in vector but instead of words we have to use senses of words. This property transforms this model from a normal word embedding to a sens embedding model.

Dataset

The dataset I used for this homework was one of the mandatory datasets, the high coverage dataset of EuroSens. The way I used it was based on the homework guidelines. Selected the english sentences and also the anchors, lemmas and the synsets in english. These are all the elements needed for creating the corpus with senses to use for training the model.

Preprocessing

As always one of the crucial parts of the homework. The preprocessing part starts with the xml parsing. As stated above I parsed the file and kept only english elements but not all of them because I dropped those lemmas, anchors and synsets which were not on the b2n_mapping.txt (the synset was not present in the list) file. I used the iterparse library for iterating the xml file and dealing with this task. Next step in preprocessing was dealing with unigram lemmas and ngram lemmas. My choice was to give priority to ngram lemmas which means that in cases when there were unigram lemmas and ngram lemmas in the same place for the same phrase I used the ngrams. After that I joined the lemmas with the corresponding synsets in the format <lemma>_<synset>. Now to transform the corpus from a normal text to a text with senses I substituted the anchors with the corresponding <lemma>_<synset> representation. But the preprocessing does not end here. I tokenized the corpus by splitting every sentence and transforming every word in lower case and the last step was to remove all the puntuacion and some buggy word like @nbps_bn:1234567n. The stop words, I kept them, because we are learning sense embeddings and stop word may change the entire context of the sentence and the context is crucial when you deal with senses.

Models

I used FastText CBOW from gensim as my final model with the following hyperparameters:
min_count = 1 (a higher min_count would reduce my number of words too much)
size = 300 (the size of vector to represent each word)
number of epoch = 60
window_size = 10
I tried to play even with other hyperparameters like negative sampling but no impact on the performance.

Experiments with the model

CBOW vs SKIP-GRAM

My model building started with the dilemma of using a CBOW or SKIP-GRAM approach. CBOW was the one used in the paper but I was not in the paper conditions because my dataset was small compared to those used by the guys in the paper and skip-gram could help me because skip-gram is better when dealing with small datasets and learns to interpret better rare words. However cbow slightly outperformed the skip-gram model (results will be shown in the table 1).

Word2Vec vs FastText

After the first dilemma I had another one to play with. It was the FastText vs Word2Vec. Well I needed a good reason to try other model given the fact that FastText is based on Word2Vec, but the good reason was there. FastText has an extra feature compared to normal Word2Vec, it treat every word as a composition of character ngrams so beside the contest it learns even the composition of the words, this characteristic can be very helpful in the case of small datasets and can also help with rare and OOV words because even if the word was not in the training corpus the combination of the character ngrams is there. And as expected , FastText outperformed Word2Vec.

Hyperparameters

As every other ML model even Word2Vec and Word2Vec based models (FastText) have hyperparametters to play and experiment with. I did some kind of manual Grid Search, by manual I intend that I tried different combination but not using some ready to use Grid Search function like sklearn GridSearchCV. The parameters that I experimented were four, number of epochs, window size, negative sampling and min_count. I did played even with the learning rate but the default one was best of those I tried. Fom my combinations I came out with the parameters I mentioned above in the model section.

Metrics and evaluation

One of the interesting points of this homework was that we could play with the metrics to use for evaluation. The one proposed by you was cosine similarity, actually that metric is one of the mos widely used in finding similarities between vectors and from different searches I made all the suggestions were about cosine but I found something interesting in theory. There is a version of cosine similarity which I think could be more useful, especially in case of nlp tasks when we are interested about context like in this case. That metric is soft cosine similarity. A soft cosine or ("soft" similarity) between two vectors considers similarities between pairs of features. With normal cosine similarity (the traditional widely used) considers the vector space model (VSM) features as independent or completely different, while the soft cosine measure proposes considering the similarity of features in VSM, which help generalize the concept of cosine (and soft cosine) as well as the idea of soft similarity. In the field of NLP the similarity among features is quite intuitive and a metric for similarity like the soft cosine might be very useful. However, to be honest I do not know how It really performs because I could not implement it on the actual task.

So finally I went with the traditional cosine similarity and implemented the word similarity task as said in the slides. I took all the synsets of pairs and for each possible combination I measured the cosine similarity with the goal of using the highest score as cosine similarity of the pair of words. After saving every similarity I used the spearman correlation to compare them with the human score from the combined.tab file.

The following table (Table 1) shows the result I got from different models during my experiments.

Spearman Correlation	Wors2Vec	FastText
CBOW	25.5	27.9
SKIP-GRAM	22.2	25.8

Table 1: FastTex vs Word2Vec with both Skip-gram and Cbow

The following table shows the hyperparameters used with the above models. Parameters are the same in FastText and Word2Vec. Because we are asked to submit only our best model in my .py file I have shown only the FastText implementation of CBOW, my final one.

Parameter	CBOW	SKIP-GRAM
Epochs	60	60
min_count	1	1
Size (embedding size)	300	300
Negative sampling	20	20
Hs (hierarchical softmax)	0	1
Sg (Skip-gram)	0	1
Window size	10	10
Workers (not important)	3	3
Learning rate	Default	Default

Tab 2: Parameters used during experiment and the best I have chosen

Now I am going to show a visualization of some similar words generated with TSNE by both Word2Vec and FastText using CBOW with the above parameters and just to recap I am using the high coverage dataset.

From the two pictures the difference is notable, in the Word2Vec case look that all senses are more similar with each other, instead in FastText the different senses and their similar words are more finely grouped and in my opinion (even from the result) FastText performs much more better than Word2Vec as it is more able to distinguish between different senses of the same word (the pictures indicate that).

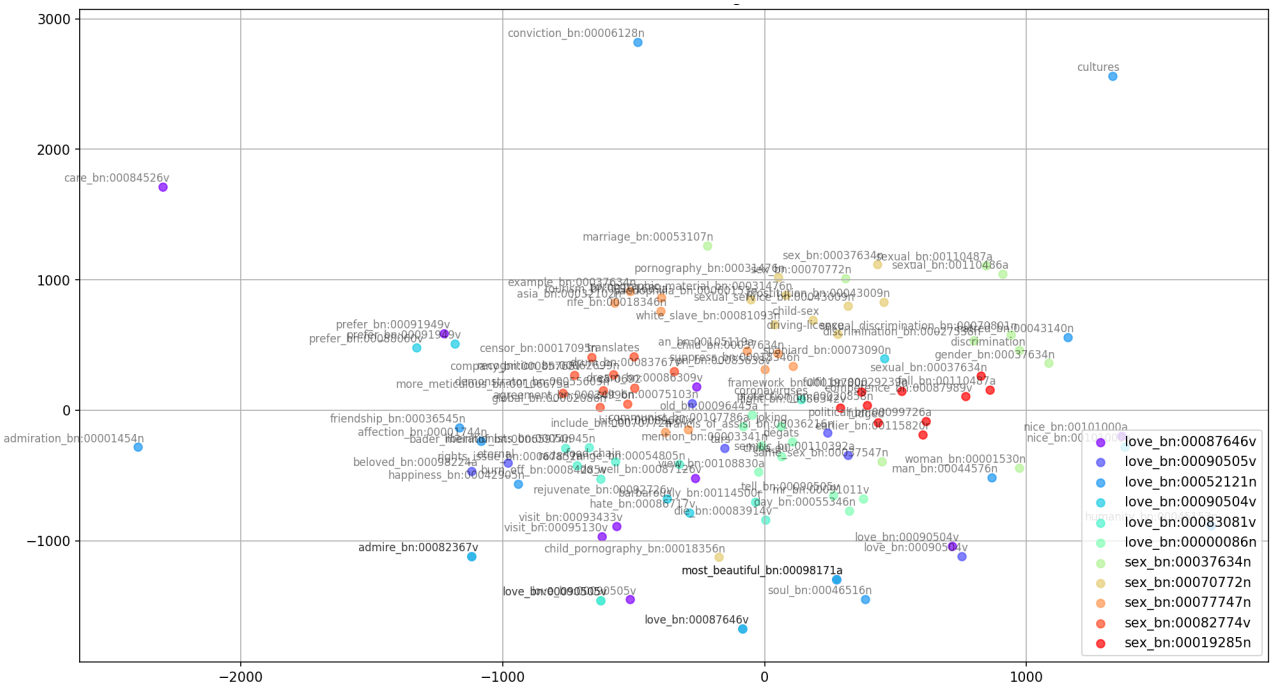

Fig 1: CBOW Word2Vec for the synsets of sex and love

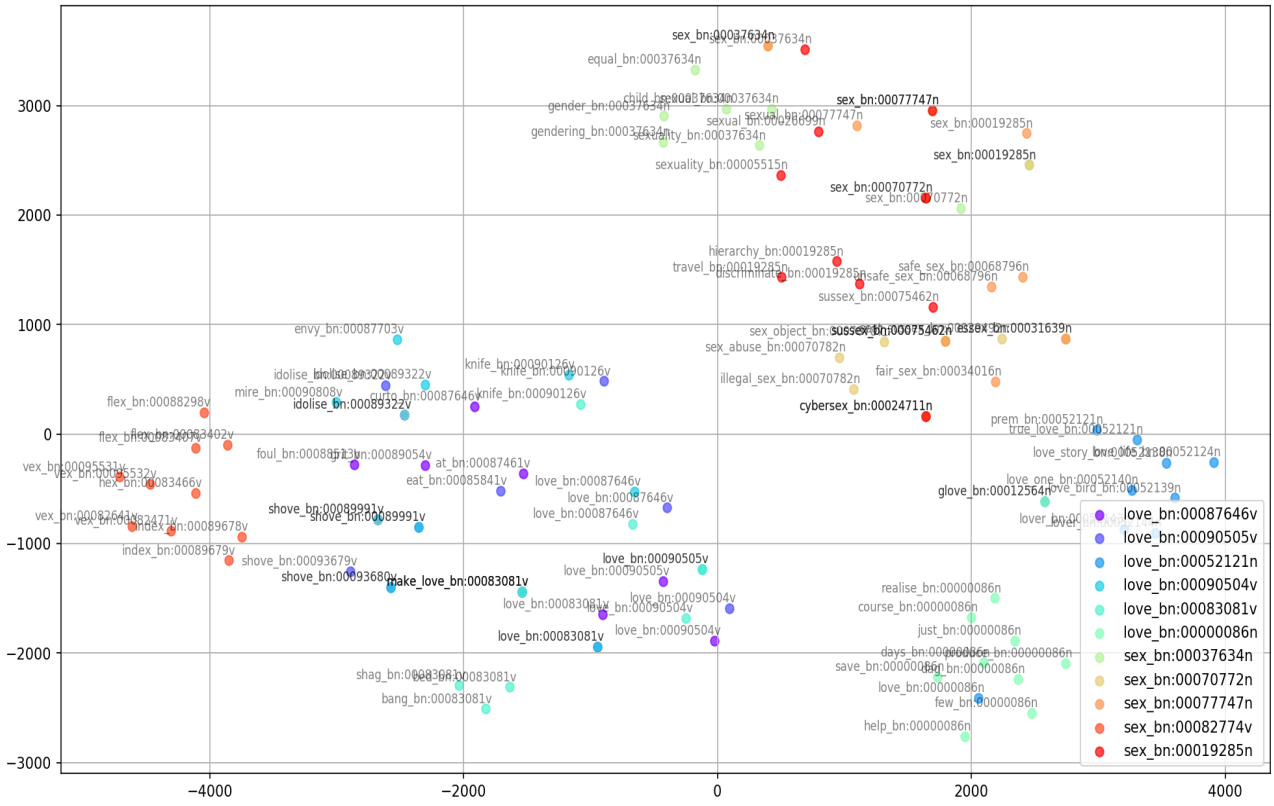

Fig. 2: CBOW FastText for the synsets of sex and love