

Distributional Word Representations Exercise

Methods in Computational Linguistics

Franziska Weeber

December 2, 2025

This exercise is based on the material of Eva Maria Vecchi.

Introduction

The main goal of this lab session is to explore word distributional representations. We will first build a word x context co-occurrence matrix from a corpus given a fixed window size. We will use this as distributional word representations. We will apply singular value decomposition to this matrix to obtain more compact, lower-dimensional representations. After that, we will compare these representations based on semantic similarity.

The participation in this lab session is voluntary: you do not have to submit any result. The work in this session should be done individually and [before the exercise session](#), but you are encouraged to discuss with your colleagues. If you want to get feedback, please ask questions during the session.

Requirements

On Ilias, you can find a python file that defines some functions and code for you to kickstart this exercise. As always, you could also start from scratch or choose a different implementation.

Data

We will use the corpora used for the corpus analysis lab session. Use more than one corpus to build these representations, so we can analyze the impact of the corpus on the word representations. These were our corpora:

- `ACL_partial_corpus.tar.gz` – a (partial) collection of ACL abstracts
- `BNCSplitWordsCorpus.tar.gz` – a dialogue corpus, part of the BNC corpus
- `english-brown.tar.gz` – a fragment of the Brown corpus (news category)
- `MovieCorpus.tar.gz` – a corpus of movie scripts
- `TwitterLowerAsciiCorpus.tar.gz` – a fragment of Twitter conversations

You should have them downloaded and ready to parse if you did the corpus analysis exercise. If not, extract them using the following command: `tar -xzf *.tar.gz`

You are also allowed – actually encouraged! – to analyze your own corpus if you have some such data that you want to work with.

Packages

We will use SciPy for singular value decomposition. It may have already been installed as a dependency last week, but you can install it into your virtual environment *venv* using the following bash script:

Listing 1: Installing Requirements

```
# EITHER activate the virtual environment (Linux, Mac)
source venv/bin/activate
# OR activate the virtual environment (Windows)
venv/Scripts/activate

# Install dependencies
pip install scipy

# Deactivate again
deactivate
```

If you do not have a virtual environment yet, you can follow the instructions in the slides for the corpus analysis exercise. You can also work without a virtual environment, but it is not recommended to avoid version conflicts.

Corpus preprocessing

You have been given functions to load the corpora. These functions are based on the solution from the Corpus Analysis exercise. You must have downloaded the solution from that exercise to the same folder as your code for this exercise so the code can be imported properly. I adapted the tokenization and token preprocessing functions the following way:

- Tokenization: Instead of returning a 1D-list of sentences and a 1D-list of tokens, this functions now returns a 2D list in which each sublist is a tokenized sentence.
- Preprocessing: Instead of returning a 1D-list of lemmas and a 1D-list of POS tags, this function now returns a 2D list in which each sublist is a tokenized sentence in which all tokens have been lemmatized and stopwords have been removed. This is supposed to standardize and reduce the context tokens and remove words whose function is purely grammatical. This function also returns a dictionary with all noun lemmas as keys and their occurrence count as value.

Distributional representation from corpora

You now have to select the target word set W . The candidates are in the lemma count dictionary from the token preprocessing function, where non-noun words and stopwords (as defined by NLTK's stopwords) have already been removed. Select all words that occur at least 50 times.

Now, you calculate the co-occurrence matrix \mathcal{M} . We choose a window size ($N = 5$), so you consider 5 words before and 5 words after your target word as context. We traverse the corpus C , and once a word w is in the target word list W , for all windows of size N centered on word w , we extract all words in this window as co-occurrences for w . On the next page, you can find pseudocode for an algorithm that you can follow, where we simulate a matrix using python dictionary structures, to be able to index on words as opposed to integers. After we build this matrix we will build the actual matrix \mathcal{M} . You can implement a different approach, by making a word index first, where you map each word to an integer, and use that as an index in a “proper” matrix. Or some other way; as always in the exercises, there is more than one correct implementation.

Singular Value Decomposition

Now, use the singular value decomposition implementation from SciPy to reduce the dimensionality of the co-occurrence matrix to 100. Use the provided function and apply it to the co-occurrence matrix \mathcal{M} before moving on to the next section.

Cosine Similarity

Now, calculate the cosine similarity between the compressed representations of all potential combinations of target word pairs.

Angular distance α between vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ is given by

$$\cos \alpha = \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\|_2 \cdot \|\mathbf{b}\|_2}$$

where $\mathbf{a}^\top \mathbf{b}$ is the matrix product of two vectors (which is equal to the dot product in this dimensionality):

$$\mathbf{a}^\top \mathbf{b} = \sum_{i=1}^n a[i] * b[i]$$

$\|\mathbf{a}\|_2$ is the L2-norm of a vector a (i. e., length):

$$\|\mathbf{a}\|_2 = \sqrt{\sum_{i=1}^n a[i]^2}$$

Finally, print the ten target word pairs with the highest cosine similarity and the ten target word pairs with the lowest cosine similarity.

Algorithm 1: Constructing a word–word co-occurrence matrix

Require: W (set of target words)
Require: C (corpus)
Require: N (window size)
Ensure: \mathcal{M} (word \times word co-occurrence matrix)

```

word_index  $\leftarrow$  empty dictionary
coocs  $\leftarrow$  empty dictionary
j  $\leftarrow$  0 (column index tracker)

for all sentence  $s \in C$  do
  sentence-tokenize line
  for  $w_t \in s$  do
    if  $w_t \in W$  then
      if  $w_t \notin \text{coocs}$  then
        coocs[ $w_t$ ]  $\leftarrow$  empty dictionary
         $\ell \leftarrow \max(0, i - N)$ 
         $r \leftarrow \min(|s| - 1, i + N)$ 
        for  $k \leftarrow \ell$  to  $r$  do
           $w_c \leftarrow s[k]$ 
          if  $w_c \neq w_t$  then
            if  $w_c \notin \text{keys}(\text{word\_index})$  then
              word_index[ $w_c$ ]  $\leftarrow$  j
              j  $\leftarrow$  j + 1
            end if
            if  $w_c \notin \text{keys}(\text{coocs}[w_t])$  then
              coocs[ $w_t$ ][ $w_c$ ]  $\leftarrow$  1
            else
              coocs[ $w_t$ ][ $w$ ]  $\leftarrow$  coocs[ $w_t$ ][ $w_c$ ] + 1
            end if
          end if
        end for
      end if
    end for
  end if
end for

```

Initialise \mathcal{M} as a $|W| \times |\text{word_index}|$ zero matrix

```

for  $i \leftarrow 0$  to  $|W| - 1$  do
   $w_t \leftarrow W[i]$ 
  for all word  $w_c$  with count  $c$  in coocs[ $w_t$ ] do
    j  $\leftarrow$  word_index[ $w_c$ ]
     $\mathcal{M}_{i,j} \leftarrow c$   $\triangleright$  (could also be relative or binary)
  end for
end for

return  $\mathcal{M}$ 

```

Input and output

W is the set of target words. C is the corpus. N is the size of the symmetric context window (N words on both sides). The result is a matrix \mathcal{M} where each row corresponds to a target word in W and each column to a context word observed in the corpus. Entries encode how often a target and a context word co-occur.

Building indices and co-occurrences

Two dictionaries are initialized. **word_index** assigns a unique column index to each context word, which will later be used to address columns in the matrix. **coocs** is another dictionary that maps context words to their co-occurrence counts for every target word.

Processing the corpus

The corpus has already been preprocessed. For every token w_t at position i in every sentence s , we check whether w_t is a target word (i. e., is in W). Only for such positions do we look at the local context.

A symmetric window of size N around i is defined. Its boundaries are adjusted so that the window does not extend beyond the sentence boundaries. Every context word w_c inside this window that is not the target word w_t is treated as a co-occurrence of word w_t .

If w_c has not been seen before as a context word, it is added to **word_index**. The pair (w_t, w_c) is then recorded in **coocs** by initializing or increasing the corresponding count.

Constructing the matrix

After the corpus has been processed, a zero-initialised matrix \mathcal{M} of size $|W| \times |\text{word_index}|$ is created. Each row corresponds to a target word in the fixed order of W , and each column corresponds to a context word in the order implied by **word_index**.

For each target word w_t we inspect its dictionary **coocs**[w_t]. For every context word w_c with count c we look up its column index j via **word_index**[w_c] and assign a value to $\mathcal{M}_{i,j}$. This value could be the absolute frequency you computed while assembling the co-occurrences, or the relative frequency, or could be just binary, or any other value – we will simply use the frequency from **coocs**. The final matrix encodes the word–word co-occurrence structure of the corpus.