SS22_NLP_embeddings

June 21, 2022

In this exercise we use a dataset of movie lines to train Word Embeddings.

1 Prepare the data

1.1 Reading the text

Load the data (see .zip file in Learnweb) into the environment of this notebook.

If you are using google colab, you can upload the file as follows:

Extract the contents of the zip file.

[1]:
```
from zipfile import ZipFile
with ZipFile('movie_lines.tsv.zip', 'r') as zipObj:
    zipObj.extractall()
```

Read the file and extract the text.

Note: reading the file with pandas or csv module does not work well, this file has a bad tsv format.

[2]:
```
movie_lines = []
for line in open('movie_lines.tsv', encoding='utf8'):
    line = line.strip()
    while line[0] == '"' and line[-1] == '"':
        line = line[1:-1]
    movie_lines.append(line.split('\t')[-1])
movie_lines = [l.replace('""', '"').replace('  ', ' ') for l in movie_lines]
len(movie_lines)
```

[2]:
```
304713
```

[3]:
```
movie_lines[50:70]
```

[3]:
```
["That's because it's such a nice one.",
 "I don't want to know how to say that though. I want to know useful things.
Like where the good stores are. How much does champagne cost? Stuff like Chat. I
have never in my life had to point out my head to someone.",
 "Right. See? You're ready for the quiz.",
 "C'esc ma tete. This is my head",
 'Let me see what I can do.',
```

```
 'Gosh if only we could find Kat a boyfriend…',
 "That's a shame.",
 'Unsolved mystery. She used to be really popular when she started high school
then it was just like she got sick of it or something.',
 'Why?',
 'Seems like she could get a date easy enough…',
 "The thing is Cameron -- I'm at the mercy of a particularly hideous breed of
loser. My sister. I can't date until she does.",
 'Cameron.',
 "No no it's my fault -- we didn't have a proper introduction ---",
 'Forget it.',
 "You're asking me out. That's so cute. What's your name again?",
 "Okay… then how 'bout we try out some French cuisine. Saturday? Night?",
 'Not the hacking and gagging and spitting part. Please.',
 "Well I thought we'd start with pronunciation if that's okay with you.",
 'Can we make this quick? Roxanne Korrine and Andrew Barrett are having an
incredibly horrendous public break- up on the quad. Again.',
 'I did.']
```

1.2 Pre-processing

Tokenization:

[4]:
```
import nltk
#nltk.download('punkt')
from nltk.tokenize import sent_tokenize
from nltk.tokenize import TweetTokenizer

tokenizer = TweetTokenizer(preserve_case=False)
```

[5]:
```
sentences = []
for line in movie_lines:
    line_sents = sent_tokenize(line)
    for sent in line_sents:
        sentences.append(tokenizer.tokenize(sent))
len(sentences)
```

[5]:
```
516218
```

[6]:
```
sentences[76:81]
```

[6]:
```
[['this', 'is', 'my', 'head'],
 ['let', 'me', 'see', 'what', 'i', 'can', 'do', '.'],
 ['gosh', 'if', 'only', 'we', 'could', 'find', 'kat', 'a', 'boyfriend', '…'],
 ["that's", 'a', 'shame', '.'],
 ['unsolved', 'mystery', '.']]
```

Further processing, e.g. removing punctuation and stopwords? While stopwords do not have lexical

meaning, we still can follow meaning for other words from their distribution around stopwords (e.g. nouns often occur after determiners; being able to distinguish words by their parts of speech can be important for semantic analyses). Thus, we should probably not remove those. If you do, you have to justify your decision and should evaluate its effect.

[7]:
```
# total number of words
len([c for clist in sentences for c in clist])
```

[7]:
```
3830690
```

[8]:
```
# number of tokens (unique words)
from collections import Counter

words = Counter(c for clist in sentences for c in clist)
len(words)
```

[8]:
```
61238
```

[9]:
```
str(words)[:154]
```

[9]:
```
"Counter({'.': 342146, 'you': 128297, '?': 110199, 'i': 103456, 'the': 99061,
'to': 80585, 'a': 71092, '-': 55093, '…': 51430, 'it': 47335, 'and': 45812,"
```

2 Training Word Embeddings

[10]:
```
# load library gensim (contains word2vec implementation)
import gensim

# ignore some warnings (probably caused by gensim version)
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.simplefilter(action='ignore', category=FutureWarning)

import multiprocessing
cores = multiprocessing.cpu_count()
```

[11]:
```
gensim.__version__
# Note: this notebook was tested with gensim version 3.6.0
```

[11]:
```
'4.2.0'
```

2.1 Initializing the NN model

Parameters: (see https://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.Word2Vec)

min_count = int - Ignores all words with total absolute frequency lower than this - (2, 100)
window = int - The maximum distance between the current and predicted word within a sentence.

E.g. window words on the left and window words on the left of our target - (2, 10) size = int - Dimensionality of the feature vectors. - (50, 300) sample = float - The threshold for configuring which higher-frequency words are randomly downsampled. Highly influencial. - (0, 1e-5) alpha = float - The initial learning rate - (0.01, 0.05) min_alpha = float - Learning rate will linearly drop to min_alpha as training progresses. To set it: alpha - (min_alpha * epochs) ~ 0.00 negative = int - If > 0, negative sampling will be used, the int for negative specifies how many "noise words" should be drown. If set to 0, no negative sampling is used. - (5, 20) workers = int - Use these many worker threads to train the model (=faster training with multicore machines) sg = {0, 1}, optional - Training algorithm: 1 for skip-gram; otherwise CBOW.

[12]:
```
import multiprocessing
cores = multiprocessing.cpu_count()

w2v_model = gensim.models.Word2Vec(min_count=5,
                                   window=3,
                                   vector_size=100,  # this parameter is called␣
 ↪'vector_size' in more recent versions of gensim
                                   workers=cores,
                                   sg=1
                                   )
```

2.2 Training the Word2Vec model with our data

[13]:
```
# defining the vocabulary based on our data
w2v_model.build_vocab(sentences, progress_per=10000)

# training
w2v_model.train(sentences, total_examples=w2v_model.corpus_count, epochs=10,␣
 ↪report_delay=1)
w2v_model.init_sims(replace=True)
```

3 Experiments with trained Word Embeddings

[14]:
```
# word vectors are stored in model.wv
print("Size of the vocabulary: %d unique words have been considered" %␣
 ↪len(w2v_model.wv.index_to_key))

example_word = 'woman'
print("\nWord vector of " + example_word)
print(w2v_model.wv[example_word])

# nearest neighbours:
print("\nWords with most similar vector representations to " + example_word)
print(w2v_model.wv.most_similar(example_word))

# similarity between two word vectors:
```

```
print("\nCosine similarity to other words:")
print(w2v_model.wv.similarity('woman', 'man'))
print(w2v_model.wv.similarity('woman', 'tree'))
```

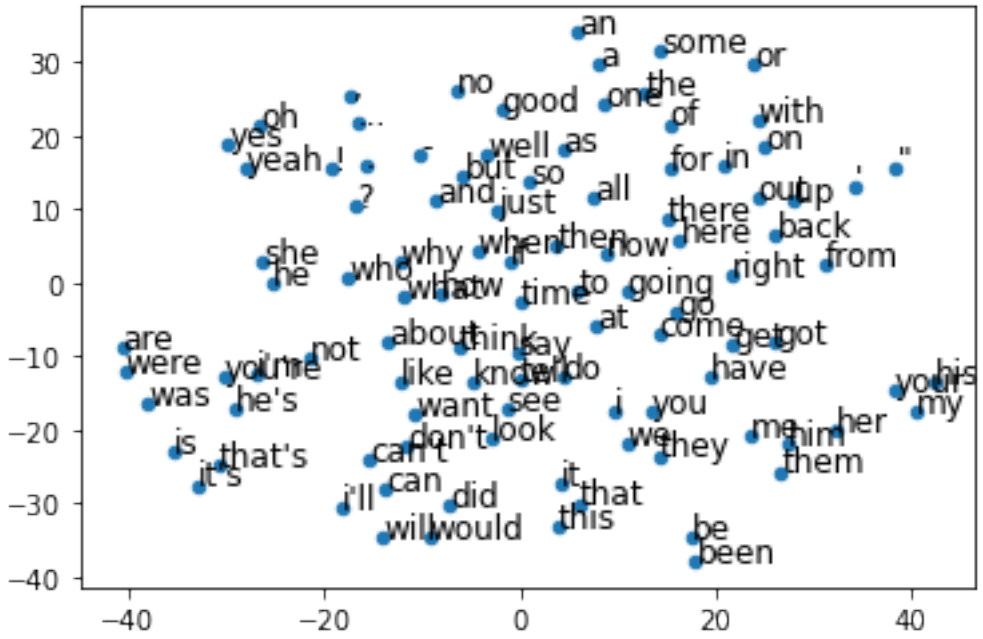
```
Size of the vocabulary: 17765 unique words have been considered

Word vector of woman
[-1.77043289e-01  1.05596259e-01 -1.08776957e-01  1.73159152e-01
  4.37706038e-02  3.41340713e-02  3.35825235e-02  3.10950670e-02
  5.74542172e-02 -2.52779275e-01 -1.21950805e-04 -1.25797793e-01
  1.03698313e-01 -3.99098508e-02  1.04754396e-01  1.23404205e-01
  1.84963979e-02 -5.37105389e-02 -4.07016370e-03 -1.79771870e-01
  2.66232435e-02  1.12816341e-01  8.91101658e-02 -3.81891802e-02
 -1.09542914e-01 -1.00102857e-01 -2.42713597e-02 -1.03569292e-01
  3.47996242e-02  1.16305113e-01  8.07844326e-02  1.19997457e-01
 -1.20469611e-02 -2.42712498e-01 -4.64356393e-02  1.52295604e-01
  1.45511389e-01 -6.20245300e-02 -4.55367379e-02 -1.09531768e-01
  1.97086707e-01 -5.15690148e-02  3.22786830e-02  4.20869701e-02
  6.56944066e-02 -1.96895399e-03 -3.08838510e-03 -1.99449748e-01
 -3.67376953e-02  4.27766442e-02  1.78877294e-01 -1.45005256e-01
 -2.09203452e-01 -3.93810049e-02  9.62130576e-02  4.11888435e-02
 -7.10374564e-02  3.84712219e-02 -1.31964669e-01  1.01733059e-01
  1.78922918e-02 -5.49210981e-02  2.63305772e-02  6.67421371e-02
 -3.40480432e-02 -5.43656759e-02 -4.16191947e-03  8.61086324e-02
  4.28840406e-02  3.11096373e-04 -4.10721600e-02 -4.66585383e-02
 -3.05782445e-02 -1.66725218e-02  1.57831550e-01 -1.38910100e-01
 -3.24376114e-02  1.40677348e-01 -3.96302901e-03  1.25381231e-01
 -1.13127440e-01 -1.33582696e-01 -2.02986568e-01  6.23893104e-02
 -1.26096432e-03  1.04858182e-01  9.72175449e-02 -2.05853079e-02
  8.90340433e-02 -3.52490358e-02  6.81220442e-02  3.21623534e-02
  5.79543300e-02 -9.40501317e-02  1.27216697e-01 -1.10028649e-03
  6.10317849e-02 -1.66947111e-01 -1.51657850e-01 -2.62054726e-02]

Words with most similar vector representations to woman
[('girl', 0.7910926342010498), ('person', 0.7394036650657654), ('man',
0.7062971591949463), ('lady', 0.6879251003265381), ('actress',
0.6855500936508179), ('gal', 0.6646643280982971), ('prostitute',
0.6638970971107483), ('child', 0.661507248878479), ('nun', 0.6549239754676819),
('creature', 0.6447154879570007)]

Cosine similarity to other words:
0.70629716
0.3715725
```

Note: The calculated nearest neighbours (mostly) seem to make sense! Remember: we did not include any knowledge database into our model; all this meaning is extracted from the occurrences in the data based on the principle of Distributional Semantics.

3.1 Visualization of word vectors

We can display (some of) the vectors in a 2d-space by reducing the dimension with PCA and t-SNE.

[15]:
```
import numpy as np
labels = []
count = 0
max_count = 100
X = np.zeros(shape=(max_count, len(w2v_model.wv['woman'])))

for term in w2v_model.wv.index_to_key:
    X[count] = w2v_model.wv[term]
    labels.append(term)
    count+= 1
    if count >= max_count: break

# It is recommended to use PCA first to reduce to ~50 dimensions
from sklearn.decomposition import PCA
pca = PCA(n_components=30)
X_50 = pca.fit_transform(X)

# Using t-SNE to further reduce to 2 dimensions
from sklearn.manifold import TSNE
model_tsne = TSNE(n_components=2, random_state=0)
Y = model_tsne.fit_transform(X_50)


# Show the scatter plot
import matplotlib.pyplot as plt
plt.scatter(Y[:,0], Y[:,1], 20)

plt.rcParams["figure.figsize"] = (20,10)

# Add labels
for label, x, y in zip(labels, Y[:, 0], Y[:, 1]):
    plt.annotate(label, xy = (x,y), xytext = (0, 0), textcoords = 'offset␣
 ↪points', size = 12)

plt.show()
```



We can also save the word vectors and look at (very cool) projections at https://projector.tensorflow.org/

[16]:
```
# save words in words.tsv
# save corresponding word embedding vectors in vectors.tsv
with open("words.tsv", 'w', encoding='utf-8') as words_outfile:
    with open("vectors.tsv", 'w', encoding='utf-8') as vectors_outfile:
        words_outfile.write('')
        vectors_outfile.write('\t'.join('' for value in w2v_model.wv[w2v_model.
↪wv.index_to_key[0]]) + '\n')
        for word in w2v_model.wv.index_to_key:
            words_outfile.write(word + '\n')
            vectors_outfile.write('\t'.join(str(value) for value in w2v_model.
↪wv[word]) + '\n')
```

3.2 Analogy recovery

[17]:
```
# we can use the parameters "positive" and "negative" to add/subtract vectors
w2v_model.wv.most_similar(positive=["woman"])
```

[17]:
```
[('girl', 0.7910926342010498),
 ('person', 0.7394036650657654),
 ('man', 0.7062971591949463),
 ('lady', 0.6879251003265381),
 ('actress', 0.6855500936508179),
 ('gal', 0.6646643280982971),
```

```
 ('prostitute', 0.6638970971107483),
 ('child', 0.661507248878479),
 ('nun', 0.6549239754676819),
 ('creature', 0.6447154879570007)]
```

"woman" is to "girl" like "man" is to ...?

"girl" - "woman" + "man"

[18]:
```
w2v_model.wv.most_similar(positive=["girl", "man"], negative=["woman"])
```

[18]:
```
[('boy', 0.7021631598472595),
 ('kid', 0.6702009439468384),
 ('gal', 0.638752818107605),
 ('guy', 0.6374076008796692),
 ('fella', 0.6318649053573608),
 ('son-of-a-bitch', 0.6253299713134766),
 ('gus', 0.6061534881591797),
 ('buddy', 0.6015918850898743),
 ('sitter', 0.5995950698852539),
 ('cat', 0.5989652872085571)]
```

[19]:
```
w2v_model.wv.most_similar(positive=["kitchen", "man"], negative=["woman"])
# note: these embeddings are trained on movie lines, which might include sexist␣
 ↪bias.
```

[19]:
```
[('locker', 0.6270676851272583),
 ('refrigerator', 0.6235319972038269),
 ('garage', 0.6204210519790649),
 ('yard', 0.6104682683944702),
 ('gym', 0.6014567017555237),
 ('cafeteria', 0.6005672216415405),
 ('fridge', 0.5879442095756531),
 ('cellar', 0.58180832862854),
 ('sled', 0.576114296913147),
 ('freeway', 0.5749862194061279)]
```

[20]:
```
w2v_model.wv.most_similar(positive=["pizza", "germany"], negative=["italy"])
# Not every attempt at analogy recovery returns results we would expect.
# This model is not perfect. The WEs are not based on that much data and the␣
 ↪dataset is biased.
```

[20]:
```
[('seafood', 0.614383339881897),
 ('straw', 0.6042576432228088),
 ('tuna', 0.597927451133728),
 ('donut', 0.5969557762145996),
 ('biscuits', 0.5960655212402344),
 ('tuba', 0.5955036282539368),
```

```
 ('iced', 0.594283938407898),
 ('low-life', 0.5896380543708801),
 ('champ', 0.5895031094551086),
 ('boiling', 0.5842174291610718)]
```

For further optimization of the training process you can use the parameter "negative" to use negative sampling instead of softmax. There is some discussion on that in the Word2Vec papers by Mikolov et al.

4 Exercises:

1. Set a reasonable value for the parameter "min_count" for training WEs on this dataset. You should always make such choices consciously and be able to explain your choice. Try to avoid "magic numbers". Give a (scientific) justification for your choice.
2. Perform a qualitative evaluation of the WE model.

```
*   Select (at least) 3 words and for each look at the nearest neighbours according to the mode
*   Define (at least) 2 experiments for analogy recovery and look at the predictions for neares
*   Select an ambiguous word - a word which you expect to have been used in movie lines with di
```

3. Remove stopwords and punctuation from the data before training Word Embeddings (you can use the lists below). Perform training and the evaluation above (2.) again. Do the results improve?

4.0.1 Question 1

I have selected the value 1 for min_count because of the reason that it will take into account more relevant words for calculating word embeddings and more words means the predictions will be more accurate.

[21]:
```
#Initializing the model with the min_count = 1
w2v_model = gensim.models.Word2Vec(min_count=1,
                                   window=3,
                                   vector_size=100,  # this parameter is called␣
 ↪'vector_size' in more recent versions of gensim
                                   workers=cores,
                                   sg=1
                                   )
```

[22]:
```
# defining the vocabulary based on data
w2v_model.build_vocab(sentences, progress_per=10000)

# training
w2v_model.train(sentences, total_examples=w2v_model.corpus_count, epochs=10,␣
 ↪report_delay=1)
w2v_model.init_sims(replace=True)
```

[23]:
```
# word vectors are stored in model.wv
print("Size of the vocabulary: %d unique words have been considered" %␣
 ↪len(w2v_model.wv.index_to_key))

example_word = 'woman'

# nearest neighbours:
print("\nWords with most similar vector representations to " + example_word)
print(w2v_model.wv.most_similar(example_word))

# similarity between two word vectors:
print("\nCosine similarity to other words:")
print(w2v_model.wv.similarity('woman', 'man'))
print(w2v_model.wv.similarity('woman', 'tree'))
```

```
Size of the vocabulary: 61238 unique words have been considered

Words with most similar vector representations to woman
[('girl', 0.8033332824707031), ('person', 0.7937257289886475), ('man',
0.7339596152305603), ('lady', 0.7198969125747681), ('prostitute',
0.7131780385971069), ('actress', 0.7129101753234863), ('nun',
0.7110179662704468), ('gal', 0.6910531520843506), ('stenographer',
0.682262659072876), ('millionaire', 0.6809649467468262)]

Cosine similarity to other words:
0.7339596
0.42401397
```

As it can be seen the cosine similarity increased when min_count was set to 1 because more words contributes for creating the word embeddings.

4.0.2 Question 2

Select (at least) 3 words and for each look at the nearest neighbours according to the model. Do the results make sense according to your idea of the meaning of the word?

[24]:
```
test_words = ['computer', 'mystery', 'game']

for wrd in test_words:
    print("\nWords with most similar vector representations to " + wrd)
    print(w2v_model.wv.most_similar(wrd))
```

```
Words with most similar vector representations to computer
[('mainframe', 0.6928532123565674), ('disc', 0.6739301681518555), ('network',
0.6687009334564209), ('fail-safe', 0.6517800092697144), ('video',
0.6475640535354614), ('archive', 0.6375060081481934), ('radio',
0.635395884513855), ('administration', 0.634968638420105), ('program',
0.628436267375946), ('codes', 0.6211806535720825)]
```

```
Words with most similar vector representations to mystery
[('democracy', 0.7545844316482544), ('wacko', 0.7365074157714844), ('wrestling',
0.7323546409606934), ('tragedy', 0.7304727435112), ('talent',
0.7223789095878601), ('adjustment', 0.7185423374176025), ('romance',
0.7173405289649963), ('profession', 0.7169383764266968), ('economy',
0.7153579592704773), ('menace', 0.7140957117080688)]

Words with most similar vector representations to game
[('poker', 0.6857396364212036), ('gig', 0.6802531480789185), ('hockey',
0.6779996156692505), ('arcade', 0.6769979596138), ('tennis',
0.6670846939086914), ('tuba', 0.6658939123153687), ('chess', 0.661096453666687),
('league', 0.6567195057868958), ('football', 0.6558771729469299), ('billiards',
0.6557881832122803)]
```

Define (at least) 2 experiments for analogy recovery and look at the predictions for nearest neighbours according to the model. Do the results make sense?

[25]:
```
w2v_model.wv.most_similar(positive=['future', 'king'])
```

[25]:
```
[('jor-el', 0.7215265035629272),
 ('quest', 0.7203720808029175),
 ('youngest', 0.7129638195037842),
 ('sanctuary', 0.7049524784088135),
 ('janitor', 0.7046830654144287),
 ('ruler', 0.701348066329956),
 ('jedi', 0.7006146311759949),
 ('legend', 0.6983488202095032),
 ('blessing', 0.6978733539581299),
 ('wessex', 0.6906274557113647)]
```

[26]:
```
w2v_model.wv.most_similar(positive=['pilot', 'army'], negative=['aeroplane'])
```

[26]:
```
[('soldiers', 0.5011719465255737),
 ('squad', 0.4884227514266968),
 ('prison', 0.47188469767570496),
 ('pilots', 0.4575333595275879),
 ('navy', 0.45265722274780273),
 ('country', 0.4493798315525055),
 ('crew', 0.44679516553878784),
 ('america', 0.44105756282806396),
 ('attorney', 0.4407457113265991),
 ('president', 0.43793249130249023)]
```

The model above make some sense but not completely because some of the predictions are not related to the outputs that we are anticipating. But if we train the model with more diverse data, the output can be improved.

Select an ambiguous word - a word which you expect to have been used in movie lines with different

meanings. What is the similarity to a related word of one meaning? What is the similarity to a related word of another/the other meaning?

[27]:
```
#The word 'head' has two meanings -> a part of body and also a Leader
ambiguous_word = 'head'
print("\nWords with most similar vector representations to " + ambiguous_word)
print(w2v_model.wv.most_similar(ambiguous_word))
```

```
Words with most similar vector representations to head
[('spine', 0.7342522144317627), ('wrists', 0.7240755558013916), ('nose',
0.7228375673294067), ('mouth', 0.711392343044281), ('belly',
0.7110751867294312), ('elbow', 0.701065182685852), ('forehead',
0.7000821828842163), ('throat', 0.6934838891029358), ('wrist',
0.6905859708786011), ('eyeballs', 0.6876295804977417)]
```

From above results, we can see that the word head is more similar to the terms related to body parts like spine, nose etc. and there are NO words that relates to leaders qualities. So we have to apply more different techniques to make it more robust.

4.0.3 Question 3

Downloading and Importing english stopwords

[28]:
```
import nltk
#nltk.download('stopwords')
from nltk.corpus import stopwords
english_stopwords = stopwords.words('english')
```

Function to remove all the punctuations from a string

[29]:
```
import string
def remove_punctuations(str_text):
    return str_text.translate(str.maketrans('', '', string.punctuation))
```

Tokenize sentences and words and removing the stop words and punctuations

[30]:
```
updated_sentences = []
for line in movie_lines:
    line_sents = sent_tokenize(remove_punctuations(line))
    for sent in line_sents:
        updated_sentences.append([word for word in tokenizer.tokenize(sent) if␣
 ↪word not in english_stopwords])
len(updated_sentences)
```

[30]: 304641

Creating the word counter

[31]:
```
words = Counter(c for clist in updated_sentences for c in clist)
len(words)
```

[31]: 65285

Creating and training the model for generating word embeddings

[32]:
```
w2v_model = gensim.models.Word2Vec(min_count=1,
                                   window=3,
                                   vector_size=100,  # this parameter is called␣
 ↪'vector_size' in more recent versions of gensim
                                   workers=cores,
                                   sg=1
                                   )
```

[33]:
```
# defining the vocabulary based on our data
w2v_model.build_vocab(updated_sentences, progress_per=10000)

# training
w2v_model.train(updated_sentences, total_examples=w2v_model.corpus_count,␣
 ↪epochs=10, report_delay=1)
w2v_model.init_sims(replace=True)
```

[34]:
```
# word vectors are stored in model.wv
print("Size of the vocabulary: %d unique words have been considered" %␣
 ↪len(w2v_model.wv.index_to_key))

example_word = 'woman'

# nearest neighbours:
print("\nWords with most similar vector representations to " + example_word)
print(w2v_model.wv.most_similar(example_word))

# similarity between two word vectors:
print("\nCosine similarity to other words:")
print(w2v_model.wv.similarity('woman', 'man'))
print(w2v_model.wv.similarity('woman', 'tree'))
```

```
Size of the vocabulary: 65285 unique words have been considered

Words with most similar vector representations to woman
[('girl', 0.6963158845901489), ('marrying', 0.6756986379623413), ('evelyn',
0.6597221493721008), ('lover', 0.6559293270111084), ('relative',
0.6483725309371948), ('madeleine', 0.6474334001541138), ('trophy',
0.6464991569519043), ('dame', 0.6462740898132324), ('lady', 0.6431680917739868),
('maxine', 0.6426957249641418)]

Cosine similarity to other words:
0.6201687
0.44141883
```

From the above, we can see that that the similarity metric decreases as we removed the stopwords

and punctuations. The stopwords provide the context of the user's intentions and provide the context to words also like word negations etc. So removing them will decrease the mappings for our word embeddings