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

Prepare the data

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:

In [1]:
```
# from google.colab import files

# uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving movie_lines.tsv.zip to movie_lines.tsv.zip
```

Extract the contents of the zip file.

In [5]:
```
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.

In [6]:
```
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)
```

Out[6]:
```
304713
```

In [7]:
```
movie_lines[1]
```

Out[7]:
```
'They do to!'
```

Pre-processing

Tokenization:

In [8]:
```
import nltk
nltk.download('punkt')
from nltk.tokenize import sent_tokenize
from nltk.tokenize import TweetTokenizer
```

```
tokenizer = TweetTokenizer(preserve_case=False)
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\simra\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [9]:
```
sentences = []
for line in movie_lines:
    line_sents = sent_tokenize(line)
    for sent in line_sents:
        sentences.append(tokenizer.tokenize(sent))
len(sentences)
```

Out[9]:
```
516218
```

In [10]:
```
sentences[8]
```

Out[10]:
```
["i'm", 'kidding', '.']
```

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.

In [11]:
```
# total number of words
len([c for clist in sentences for c in clist])
```

Out[11]:
```
3830690
```

In [12]:
```
# number of tokens (unique words)
from collections import Counter

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

Out[12]:
```
61238
```

In [13]:
```
str(words)[:154]
```

Out[13]:
```
"Counter({'.': 342146, 'you': 128297, '?': 110199, 'i': 103456, 'the': 99061, 'to':
80585, 'a': 71092, '-': 55093, '...': 51430, 'it': 47335, 'and': 45812,"
```

Training Word Embeddings

In [14]:
```
# 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()
```

In [15]:
```
gensim.__version__
# Note: this notebook was tested with gensim version 3.6.0
```

Out[15]:
```
'4.2.0'
```

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.

In [16]:
```
import multiprocessing
cores = multiprocessing.cpu_count()

w2v_model = gensim.models.Word2Vec(min_count=5,
                                   window=3,
                                   vector_size=100,  # this parameter is called 'vec
                                   workers=cores,
                                   sg=1
                                   )
```

Training the Word2Vec model with our data

In [17]:
```
# 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_
w2v_model.init_sims(replace=True)
```

Experiments with trained Word Embeddings

In [18]:
```
# word vectors are stored in model.wv
print("Size of the vocabulary: %d unique words have been considered" % len(w2v_model

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
[-0.07723571  0.09369257 -0.00196505  0.05918895  0.11202592 -0.06087609
 -0.03536097  0.19769712  0.04461017 -0.31839883  0.16567387  0.05571393
  0.05443341  0.01414167  0.1118309   0.04945192  0.10689943 -0.00371925
  0.02442241 -0.13684697  0.01497499  0.20631053 -0.00672911  0.00670008
 -0.07456276  0.02587368 -0.09102437  0.00716512  0.0134811   0.10238875
  0.13993882  0.07611352 -0.00301115 -0.24410237 -0.05930375  0.10752871
  0.0573089   0.05094322  0.01962757  0.0017972   0.13359176 -0.18569905
  0.07165518 -0.03694989  0.09805827 -0.03119434  0.03695795 -0.14485358
 -0.07048575 -0.0294713   0.2308051  -0.11914063 -0.16862893  0.03612529
  0.24808507  0.11133115  0.02790032  0.11412625 -0.05065009  0.01076042
  0.09486009 -0.07988816 -0.08114596  0.03598337 -0.12864357  0.1679399
  0.08094576  0.20646743  0.09950966 -0.0228746   0.01071747  0.00868971
 -0.00183441 -0.0523626   0.07844878 -0.02571374 -0.0266796  -0.04982262
 -0.09288929  0.06863582 -0.06141902 -0.05938187 -0.12194014 -0.00145506
 -0.05089483  0.02359966  0.06257191 -0.00794642 -0.02323873 -0.12715767
  0.06940137 -0.02485243  0.1020613   0.09272288  0.22387117 -0.05259477
 -0.02566241 -0.06034735  0.0059331  -0.03344933]

Words with most similar vector representations to woman
[('girl', 0.7846516966819763), ('person', 0.748774528503418), ('man', 0.709419131278
9917), ('lady', 0.6730239391326904), ('actress', 0.6681362986564636), ('child', 0.66
37682914733887), ('creature', 0.6592048406600952), ('chick', 0.651640772819519), ('s
tenographer', 0.6506122350692749), ('prostitute', 0.6366931200027466)]

Cosine similarity to other words:
0.709419
0.34603578
```

Note:

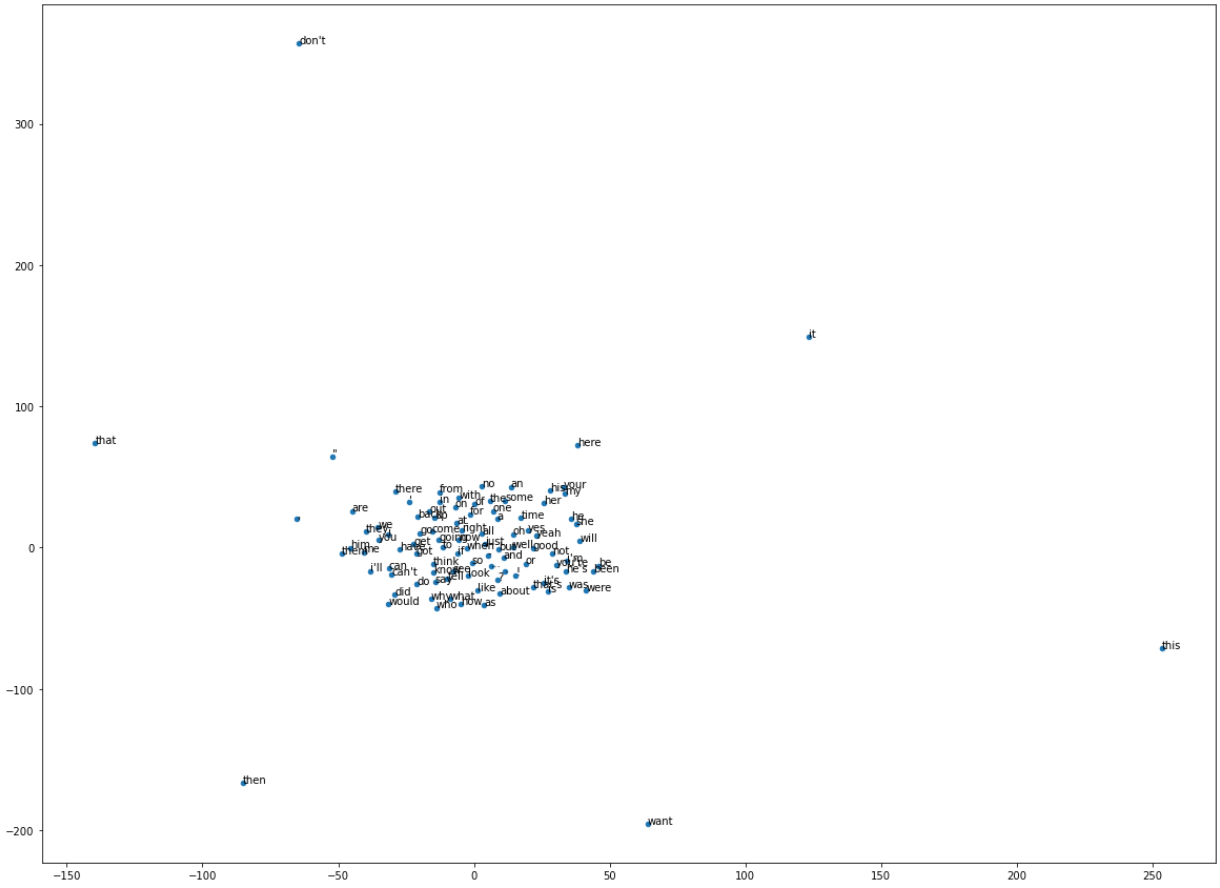
The calculated nearest neighbours (mostly) seem to make sense! Remember: we did not include any knowledge database into our model; all this meaning is extracted from the occurrences in the data based on the principle of Distributional Semantics.

Visualization of word vectors

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

In [20]:
```
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.key_to_index:
    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"] = (20,15)

# 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', s

plt.show()
```



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

In [21]:
```
# save words in words.tsv
# save corresponding word embedding vectors in vectors.tsv
with open("words.tsv", 'w', encoding = 'utf8') as words_outfile:
    with open("vectors.tsv", 'w', encoding = 'utf8') as vectors_outfile:
        words_outfile.write('word\n')
        for word in w2v_model.wv.key_to_index:
            words_outfile.write(word + "\n")
            vectors_outfile.write('\t'.join(str(value) for value in w2v_model.wv[wor
```

Analogy recovery

In [22]:
```
# we can use the parameters "positive" and "negative" to add/subtract vectors
w2v_model.wv.most_similar(positive=["woman"])
```

Out[22]:
```
[('girl', 0.7846516966819763),
 ('person', 0.748774528503418),
 ('man', 0.7094191312789917),
 ('lady', 0.6730239391326904),
 ('actress', 0.6681362986564636),
 ('child', 0.6637682914733887),
 ('creature', 0.6592048406600952),
 ('chick', 0.651640772819519),
 ('stenographer', 0.6506122350692749),
 ('prostitute', 0.6366931200027466)]
```

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

"girl" - "woman" + "man"

In [23]:
```
w2v_model.wv.most_similar(positive=["girl", "man"], negative=["woman"])
```

Out[23]:
```
[('boy', 0.7278891801834106),
 ('gal', 0.6680917143821716),
 ('guy', 0.6527993083000183),
 ('kid', 0.6300188302993774),
 ('fella', 0.6104516386985779),
 ('son-of-a-bitch', 0.6099401712417603),
 ('policeman', 0.5861771702766418),
 ('buddy', 0.5831032395362854),
 ('lenny', 0.5808266997337341),
 ('fellow', 0.5794304609298706)]
```

In [24]:
```
w2v_model.wv.most_similar(positive=["kitchen", "man"], negative=["woman"])
# note: these embeddings are trained on movie lines, which might include sexist bias
```

Out[24]:
```
[('garage', 0.6584079265594482),
 ('yard', 0.6449903845787048),
 ('cellar', 0.642348051071167),
 ('locker', 0.613688051700592),
 ('pub', 0.6119754314422607),
 ('cafeteria', 0.6072391271591187),
 ('fridge', 0.6014037728309631),
 ('gym', 0.5938693284988403),
 ('cupboard', 0.5936901569366455),
 ('shovel', 0.5887446403503418)]
```

In [25]:
```
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
```

Out[25]:
```
[('scoop', 0.5998009443283081),
 ('tequila', 0.5941002368927002),
 ('compass', 0.5845178961753845),
 ('soda', 0.5844212770462036),
 ('bark', 0.5834405422210693),
 ('sack', 0.5829185247421265),
 ('weed', 0.5799546837806702),
 ('low-life', 0.5792235136032104),
 ('sip', 0.5774196982383728),
 ('mechanic', 0.5761393308639526)]
```

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.

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 model. Do the results make sense according to your idea of the meaning of the word?
   - 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?
   - 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? (for example if you would select "mouse", you could compute the similarity to "rat" and to "keyboard" and would expect both values to be relatively high)

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?

In [26]:
```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "yo
u've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'h
is', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itse
lf', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'who
m', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were',
'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing',
'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of',
'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'durin
g', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'o
n', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'wh
en', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'ot
her', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'to
```

```
o', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've",
'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "cou
ldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'h
aven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'n
eedn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'were
n', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\simra\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [27]:
```
import string
string.punctuation
```

Out[27]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

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".

On testing with both lower and higher min_count, I have seen that lower min_count gives more closely related nearest neighbourhoods, also for analogy sometimes the less frequent words are required.

In [28]:
```
w2v_model_2 = gensim.models.Word2Vec(min_count=2,
                                     window=3,
                                     vector_size=100,  # this parameter is called 'vec
                                     workers=cores,
                                     sg=1
                                     )
```

In [29]:
```
# defining the vocabulary based on our data
w2v_model_2.build_vocab(sentences, progress_per=10000)

# training
w2v_model_2.train(sentences, total_examples=w2v_model_2.corpus_count, epochs=10, rep
w2v_model_2.init_sims(replace=True)
```

1. Perform a qualitative evaluation of the WE model.

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?

For the word friend, the nearest neighbourhoods are closely related whereas for the word joy, most of the words carry opposite meaning this may be due to some negative words in front of them.

In [30]:
```
word1 = 'friend'
word2 = 'joy'
word3 = 'busy'
```

In [31]:
```
print("\nWords with most similar vector representations to " + word1)
```

```
print(w2v_model_2.wv.most_similar(word1))
```

```
Words with most similar vector representations to friend
[('partner', 0.7232656478881836), ('son-in-law', 0.672023355960846), ('hairdresser',
0.665221631526947), ('brother', 0.6646710634231567), ('lover', 0.658385157585144),
('colleague', 0.6534223556518555), ('friends', 0.6456687450408936), ('cousin', 0.645
4216241836548), ('frienda', 0.6436011791229248), ('servant', 0.6394617557525635)]
```

In [32]:
```
print("\nWords with most similar vector representations to " + word2)
print(w2v_model_2.wv.most_similar(word2))
```

```
Words with most similar vector representations to joy
[('happiness', 0.6874597072601318), ('remorse', 0.6852285265922546), ('hesitation',
0.6820386052131653), ('meaner', 0.6815669536590576), ('loneliness', 0.68128597736358
64), ('lust', 0.6787198185920715), ('kindness', 0.6773571968078613), ('pace', 0.6731
580495834351), ('dignity', 0.6701396703720093), ('butterflies', 0.6690418720245361)]
```

In [33]:
```
print("\nWords with most similar vector representations to " + word3)
print(w2v_model_2.wv.most_similar(word3))
```

```
Words with most similar vector representations to busy
[('chipper', 0.6826261281967163), ('impatient', 0.6678988337516785), ('tense', 0.662
0700359344482), ('discouraged', 0.6483463644981384), ('grounded', 0.647227883338928
2), ('distracting', 0.645234227180481), ('excited', 0.6350381374359131), ('chilly',
0.6268269419670105), ('disgusted', 0.6238229274749756), ('noisy', 0.619710624217987
1)]
```

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?

I have taken the word friend and it's positive similarity with man and negative with woman and vice versa.

In [39]:
```
w2v_model_2.wv.most_similar(positive=[word1, "man"], negative=["woman"])
```

Out[39]:
```
[('partner', 0.6586190462112427),
 ('buddy', 0.5971649885177612),
 ('brother', 0.588058352470398),
 ('friends', 0.5749609470367432),
 ('hairdresser', 0.5589945912361145),
 ('uncle', 0.5438713431358337),
 ('colleague', 0.5407683849334717),
 ('bernie', 0.5329256057739258),
 ('son', 0.5264003276824951),
 ('larry', 0.5220411419868469)]
```

In [35]:
```
w2v_model_2.wv.most_similar(positive=[word1, "woman"], negative=["man"])
```

Out[35]:
```
[('sister', 0.6266343593597412),
 ('daughter', 0.6167484521865845),
 ('roommate', 0.6108320951461792),
 ('husband', 0.5977205038070679),
 ('companion', 0.5951350927352905),
 ('girlfriend', 0.5921329259872437),
 ('wife', 0.5791374444961548),
 ('girl', 0.5727143883705139),
 ('mother', 0.5714119672775269),
 ('grandmother', 0.567990779876709)]
```

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? (for example if you would select "mouse", you could compute the similarity to "rat" and to "keyboard" and would expect both values to be relatively high)

The word radius in the document has two meanings - circle radius and distance. It makes more sense with the distance in context with the document.

In [36]:
```
print("\nCosine similarity to other words:")
print(w2v_model_2.wv.similarity('radius', 'circle'))
print(w2v_model_2.wv.similarity('radius', 'mile'))
```

```
Cosine similarity to other words:
0.53733057
0.73443675
```

1. 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?

Performing the same experiment after removing stopwards and punctuations.

In [37]:
```
sents = []
for line in movie_lines:
    line2 = line.translate(str.maketrans('', '', string.punctuation))  # removing pu
    line_sents = sent_tokenize(line2)
    for sent in line_sents:
        sents.append(tokenizer.tokenize(sent))

len(sents)
```

Out[37]:
```
304641
```

In [40]:
```
sents[0]
```

Out[40]:
```
['they', 'do', 'not']
```

In [41]:
```
for sent in sents:
    for word in sent:
        if word in stopwords.words('english'):     # removing stopwords
            sent.remove(word)
```

In [42]:
```
w2v_model_ = gensim.models.Word2Vec(min_count=2,
                                    window=3,
                                    vector_size=100,  # this parameter is called 'vec
                                    workers=cores,
                                    sg=1)
```

In [43]:
```
# defining the vocabulary based on our data
w2v_model_.build_vocab(sents, progress_per=10000)
```

```
# training
w2v_model_.train(sents, total_examples=w2v_model_.corpus_count, epochs=10, report_de
w2v_model_.init_sims(replace=True)
```

Finding nearest neighbourhoods on the same data after removing stopwords and punctuations tends to give more closely related words in the case of friend and joy whereas 'busy' doesnot show much improvement.

In [44]:
```
word1 = 'friend'
word2 = 'joy'
word3 = 'busy'
```

In [45]:
```
print("\nWords with most similar vector representations to " + word1)
print(w2v_model_.wv.most_similar(word1))
```

```
Words with most similar vector representations to friend
[('johana', 0.6641167402267456), ('friends', 0.6618260145187378), ('colleague', 0.64
80444669723511), ('associate', 0.6450671553611755), ('paden', 0.6315784454345703),
('frienda', 0.6310584545135498), ('stepmother', 0.6245312690734863), ('sheshe', 0.62
007737159729), ('eleanor', 0.6133481860160828), ('acquaintance', 0.612666845321655
3)]
```

In [46]:
```
print("\nWords with most similar vector representations to " + word2)
print(w2v_model_.wv.most_similar(word2))
```

```
Words with most similar vector representations to joy
[('lust', 0.7507855892181396), ('doth', 0.7442662715911865), ('eternal', 0.730460464
9543762), ('devotion', 0.7221009731292725), ('defy', 0.7200002670288086), ('didst',
0.705913245677948), ('cynicism', 0.6974148750305176), ('warmth', 0.694862127304077
1), ('happiness', 0.6924903392791748), ('cherish', 0.6895419359207153)]
```

In [47]:
```
print("\nWords with most similar vector representations to " + word3)
print(w2v_model_.wv.most_similar(word3))
```

```
Words with most similar vector representations to busy
[('excited', 0.6731618046760559), ('quitting', 0.6636108756065369), ('chilly', 0.662
664532661438), ('explaining', 0.6590968370437622), ('chatting', 0.658191978931427),
('stayin', 0.6538291573524475), ('chipper', 0.6527941226959229), ('autopilot', 0.647
3845839500427), ('risky', 0.6448237895965576), ('exaggerating', 0.6400189995765686)]
```

In [48]:
```
w2v_model_.wv.most_similar(positive=[word1, "man"], negative=["woman"])
```

Out[48]:
```
[('buddy', 0.6203075647354126),
 ('jack', 0.5708543062210083),
 ('lucas', 0.5699402689933777),
 ('partner', 0.5696719884872437),
 ('friends', 0.5693548321723938),
 ('pal', 0.5618534088134766),
 ('bernie', 0.5597748160362244),
 ('jenkins', 0.542865514755249),
 ('roderick', 0.5319476127624512),
 ('stevens', 0.531903862953186)]
```

In [49]:
```
w2v_model_.wv.most_similar(positive=[word1, "woman"], negative=["man"])
```

Out[49]:
```
[('sister', 0.56409752368927),
 ('faithful', 0.5594152212142944),
 ('daughter', 0.5557692050933838),
 ('girlfriend', 0.5473512411117554),
 ('recluse', 0.5331935286521912),
 ('boyfriend', 0.532207727432251),
 ('marylin', 0.530866801738739),
 ('editor', 0.5249834060668945),
 ('antoinette', 0.5239260196685791),
 ('stepmother', 0.5236356258392334)]
```

In [50]:
```
print("\nCosine similarity to other words:")
print(w2v_model_.wv.similarity('radius', 'circle'))
print(w2v_model_.wv.similarity('radius', 'mile'))
```

```
Cosine similarity to other words:
0.47661865
0.7451074
```

In [ ]: