SI630 Homework 2: Word2vec Vector Analysis (For Part 3)

Important Note: Start this notebook only after you've gotten your word2vec model up and running!

Many NLP packages support working with word embeddings. In this notebook you can work through the various problems assigned in Task 3. We've provided the basic functionality for loading word vectors using Gensim (https://radimrehurek.com/gensim/models/keyedvectors.html), a good library for learning and using word vectors, and for working with the vectors.

One of the fun parts of word vectors is getting a sense of what they learned. Feel free to explore the vectors here!

In [88]:
```
import pandas as pd

from gensim.models import KeyedVectors
from gensim.test.utils import datapath
```

In [89]:
```
model_name = "third_model"
word_vectors = KeyedVectors.load_word2vec_format('./models/'+model_name+'.kv', binary=False)
```

In [90]:
```
word_vectors['the']
```

```
array([-0.803966  ,  0.20739706, -0.41470504, -0.09618309,  0.15361309,
        0.36841783, -0.21240279,  0.01897331, -0.09516799, -0.29754043,
       -0.27828205, -0.05730286,  0.08532567,  0.191788  , -0.6630302 ,
       -0.37582618,  0.17940123, -0.43394163, -0.59503525,  0.77354383,
       -0.4571756 ,  0.0418206 ,  0.4222303 , -0.24021253,  0.2567409 ,
        0.18184762,  0.2423835 ,  0.13162646,  0.14234862, -0.13687187,
        0.06895836,  0.3538237 ,  0.01597746, -0.14195429, -0.40593222,
       -0.47744456, -0.19720761,  0.21820286,  0.2837015 ,  0.24548618,
        0.2045187 , -0.63596004, -0.03282333,  0.4831505 , -0.44626796,
        0.13517477,  0.2952347 , -0.0601838 , -0.31692913, -0.5239965 ],
      dtype=float32)
```

In [91]:
```
word_vectors.similar_by_word("books")
```

```
[('articles', 0.9607430100440979),
 ('papers', 0.9235630035400391),
 ('novels', 0.9147697687149048),
 ('poems', 0.9087007641792297),
 ('chapters', 0.9040026664733887),
 ('monographs', 0.9021276235580444),
 ('works', 0.9005582332611084),
 ('essays', 0.8988243341445923),
 ('publications', 0.8987964391708374),
 ('stories', 0.8964657187461853)]
```

In [92]:
```
def get_analogy(a, b, c):
    return word_vectors.most_similar(positive=[b, c], negative=[a])[0][0]
```

In [93]:
```
get_analogy('man', 'woman', 'king')
```

```
'queen'
```

Problem 15

In [94]:
```
word_list = [
    "bread",
    "physics",
    "encourage",
    "revenue",
    "covid",
    "genetic",
    "supernova",
    "oxidation",
    "serendipity",
    "concession"
]

most_similar_word_list = []

for word in word_list:
    most_similar_word_list.append(word_vectors.similar_by_word(word)[0][0])

word_df = pd.DataFrame({
    "word": word_list,
    "similar word": most_similar_word_list
})

word_df
```

	word	similar word
0	bread	vegetables
1	physics	chemistry
2	encourage	enhance
3	revenue	tax
4	covid	pandemic
5	genetic	protein
6	supernova	gently
7	oxidation	stucco
8	serendipity	weeting
9	concession	debtors

From the table shown above (from the most common words to rare ones), we can find out the facts that:

- For those commonly used words (0 - 6), we can easily figure out the relationship between the target word and its most similar word. For example, physics and chemistry are both categoried as natural sciences.
- However, for those rare words, we can see that the model works not so well on them like connecting the oxidation with stucco. We can hardly figure out relationship in bwtween. It may results from the lack of training examples.

(The whole list of similar words for each target word are shown for your reference)

In [95]:
```
word_vectors.similar_by_word("bread")
```

```
[('vegetables', 0.9718700647354126),
 ('toxic', 0.970106840133667),
 ('acrylic', 0.9676809906959534),
 ('molecules', 0.9675472974777222),
 ('crude', 0.9674468636512756),
 ('healing', 0.9667180776596069),
 ('gravity', 0.9660497307777405),
 ('liquid', 0.9651873707771301),
 ('proteins', 0.9649662375450134),
 ('humans', 0.9645667672157288)]
```

In [96]:
```
word_vectors.similar_by_word("physics")
```

```
[('chemistry', 0.992280125617981),
 ('biology', 0.9870692491531372),
 ('engineering', 0.981624960899353),
 ('mathematics', 0.9779875874519348),
 ('economics', 0.9759252667427063),
 ('physiology', 0.9699663519859314),
 ('psychology', 0.9684190154075623),
 ('geology', 0.9681183099746704),
 ('sociology', 0.9652984738349915),
 ('electrical', 0.962410569190979)]
```

In [97]:
```
word_vectors.similar_by_word("encourage")
```

```
[('enhance', 0.9857335686683655),
 ('analyze', 0.9803969264030457),
 ('aims', 0.9803727269172668),
 ('generate', 0.9786515235900879),
 ('encouraging', 0.9777085185050964),
 ('integrate', 0.9749067425727844),
 ('preserve', 0.9743477702140808),
 ('engage', 0.9742167592048645),
 ('introduce', 0.9726227521896362),
 ('explore', 0.9710420966148376)]
```

In [98]:
```
word_vectors.similar_by_word("revenue")
```

```
[('tax', 0.953637421131134),
 ('expenditures', 0.9432281255722046),
 ('aircraft', 0.941349446773529),
 ('statewide', 0.9402140974998474),
 ('implementation', 0.9333816766738892),
 ('arbitration', 0.9323774576187134),
 ('units', 0.9320067167282104),
 ('resolution', 0.9312455654144287),
 ('expenditure', 0.9311120510101318),
 ('task', 0.9306795597076416)]
```

In [99]:
```
word_vectors.similar_by_word("covid")
```

```
[('pandemic', 0.9395113587379456),
 ('coronavirus', 0.9303358197212219),
 ('deadline', 0.8798041343688965),
 ('expire', 0.8621783256530762),
 ('hayemaker', 0.8589552640914917),
 ('nitro', 0.8506273031234741),
 ('himara', 0.84943687915802),
 ('ttff', 0.8483490943908691),
 ('pipa', 0.8479142785072327),
 ('tuesday', 0.8475530743598938)]
```

In [100]:
```
word_vectors.similar_by_word("genetic")
```

```
[('protein', 0.9793424606323242),
 ('linear', 0.9696540832519531),
 ('functional', 0.9683409929275513),
 ('mechanisms', 0.9682909846305847),
 ('neural', 0.9656184315681458),
 ('materials', 0.9642196297645569),
 ('devices', 0.9641343951225281),
 ('processes', 0.9639464020729065),
 ('fluid', 0.9637303352355957),
 ('differential', 0.961711049079895)]
```

In [101]:
```
word_vectors.similar_by_word("supernova")
```

```
[('gently', 0.9779091477394104),
 ('silly', 0.9743908047676086),
 ('fancy', 0.9738282561302185),
 ('foolish', 0.9699468612670898),
 ('donkey', 0.9692879915237427),
 ('accents', 0.9675639271736145),
 ('sincerity', 0.9652719497680664),
 ('phrases', 0.9646726250648499),
 ('melt', 0.9645059108734131),
 ('masculine', 0.9643701910972595)]
```

In [102]:
```
word_vectors.similar_by_word("oxidation")
```

```
[('stucco', 0.9436200261116028),
 ('rectifier', 0.9348612427711487),
 ('ᴛ', 0.9347659945487976),
 ('individuation', 0.9324919581413269),
 ('semigroups', 0.9321315288543701),
 ('prophetic', 0.9318394064903259),
 ('transformer', 0.9317874908447266),
 ('denoting', 0.9300829768180847),
 ('horizontal', 0.930073082447052),
 ('deformation', 0.9283044338226318)]
```

In [103]:
```
word_vectors.similar_by_word("serendipity")
```

```
[('weeting', 0.98467618227005),
 ('woz', 0.983461320400238),
 ('ableton', 0.9830171465873718),
 ('bhavana', 0.9825843572616577),
 ('lukens', 0.9821453094482422),
 ('preethi', 0.9817216992378235),
 ('gecenin', 0.9816479682922363),
 ('cantus', 0.9809957146644592),
 ('tn5', 0.9809512495994568),
 ('kelebek', 0.9806206226348877)]
```

In [104]:
```
word_vectors.similar_by_word("concession")
```

```
[('debtors', 0.9847534894943237),
 ('rioting', 0.9748736023902893),
 ('queue', 0.9741414785385132),
 ('occupancy', 0.9740566611289978),
 ('uninhabited', 0.9738625884056091),
 ('modernise', 0.9724675416946411),
 ('hispania', 0.9724375605583191),
 ('expressly', 0.972412109375),
 ('lithuanians', 0.9723716974258423),
 ('elephantine', 0.9722658395767212)]
```

Problem 16

In [116]:
```
def print_get_analogy(a, b, c):
    print(f"{b} - {a} + {c} = {get_analogy(a, b, c)}")

# print_get_analogy('man', 'king', 'woman') # example
print_get_analogy('attack', 'sword', 'defense')
print_get_analogy('up', 'jump', 'down')
print_get_analogy('liquid', 'water', 'solid')
print_get_analogy('one', 'january', 'six')
print_get_analogy('human', 'earth', 'sun')
```

```
sword - attack + defense = non
jump - up + down = pistol
water - liquid + solid = wide
january - one + six = june
earth - human + sun = devil
```

According to the five examples above, we can see that not all equations work the same as my thought (for the first 3 examples). For example, I orginally think that jump - up + down will lead to a "fall". However, it does not work. But for the last two examples, it makes sense in to some extent.

From my point of view, it is because the model is not trained well enough to explore the similarity in between. We may need more training data and further training to reach a better result.

In [ ]:
```
```