SemAxis is a method for scoring terms along a user-defined axis (e.g., positive-negative, concrete-abstract, hot-cold), which can be used for a range of empirical questions (for one example, see Kozlowski et al. 2019). In this homework, you'll implement SemAxis using word representations from Glove, and use it to explore corpus-specific conceptual associations.

Before running, install gensim with:

```
conda install gensim
```

In [34]:

```
import re
from gensim.models import KeyedVectors
from gensim.scripts.glove2word2vec import glove2word2vec
import numpy as np
import numpy.linalg as LA
```

In this homework, we'll be working with pre-trained word embeddings using the `gensim` library, which provides a number of functions for accessing representations for individual words and comparing them. The representations we'll use come from Glove, which are trained on web data from the Common Crawl corpus.

In [35]:

```
# First we have to convert the Glove format into w2v format; this creates a
glove_file="../data/glove.6B.100d.100K.txt"
glove_in_w2v_format="../data/glove.6B.100d.100K.w2v.txt"
_ = glove2word2vec(glove_file, glove_in_w2v_format)
```

```
/var/folders/b2/f5_sk7l16f1cr1htyq1k3c5c0000gn/T/ipykernel_25144/2636338883
.py:4: DeprecationWarning: Call to deprecated `glove2word2vec` (KeyedVector
s.load_word2vec_format(.., binary=False, no_header=True) loads GLoVE text v
ectors.).
  _ = glove2word2vec(glove_file, glove_in_w2v_format)
```

In [36]:

```
glove = KeyedVectors.load_word2vec_format("../data/glove.6B.100d.100K.w2v.t
```

In [37]:

```
good_vector=glove["good"]
```

In [38]:

```
print(good_vector)
```

```
[-0.030769   0.11993    0.53909   -0.43696   -0.73937   -0.15345
  0.081126  -0.38559   -0.68797   -0.41632   -0.13183   -0.24922
  0.441      0.085919   0.20871   -0.063582   0.062228  -0.051234
 -0.13398    1.1418     0.036526   0.49029   -0.24567   -0.412
  0.12349    0.41336   -0.48397   -0.54243   -0.27787   -0.26015
 -0.38485    0.78656    0.1023    -0.20712    0.40751    0.32026
 -0.51052    0.48362   -0.0099498 -0.38685    0.034975  -0.167
  0.4237    -0.54164   -0.30323   -0.36983    0.082836  -0.52538
 -0.064531  -1.398     -0.14873   -0.35327   -0.1118     1.0912
  0.095864  -2.8129     0.45238    0.46213    1.6012    -0.20837
 -0.27377    0.71197   -1.0754    -0.046974   0.67479   -0.065839
  0.75824    0.39405    0.15507   -0.64719    0.32796   -0.031748
  0.52899   -0.43886    0.67405    0.42136   -0.11981   -0.21777
 -0.29756   -0.1351     0.59898    0.46529   -0.58258   -0.02323
 -1.5442     0.01901   -0.015877   0.024499  -0.58017   -0.67659
 -0.040379  -0.44043    0.083292   0.20035   -0.75499    0.16918
 -0.26573   -0.52878    0.17584    1.065    ]
```

Functions useful for the first question include the following:

In [39]:

```
# access the representation for a single word
great_vector=glove["great"]
print(great_vector)
# use numpy to average multiple vector representations together
vecs_to_average=[good_vector, great_vector]
average=np.mean(vecs_to_average, axis=0)

# calculate the cosine similariy between two vectors
cosine_similarity=glove.cosine_similarities(good_vector, [great_vector])

print(good_vector.shape, great_vector.shape, average.shape, cosine_similar:
```

```
[-0.013786   0.38216    0.53236    0.15261   -0.29694   -0.20558
 -0.41846   -0.58437   -0.77355   -0.87866   -0.37858   -0.18516
 -0.128     -0.20584   -0.22925   -0.42599    0.3725     0.26077
 -1.0702     0.62916   -0.091469   0.70348   -0.4973    -0.77691
  0.66045    0.09465   -0.44893    0.018917   0.33146   -0.35022
 -0.35789    0.030313   0.22253   -0.23236   -0.19719   -0.0053125
 -0.25848    0.58081   -0.10705   -0.17845   -0.16206    0.087086
  0.63029   -0.76649    0.51619    0.14073    1.019     -0.43136
  0.46138   -0.43585   -0.47568    0.19226    0.36065    0.78987
  0.088945  -2.7814    -0.15366    0.01015    1.1798     0.15168
 -0.050112   1.2626    -0.77527    0.36031    0.95761   -0.11385
  0.28035   -0.02591    0.31246   -0.15424    0.3778    -0.13599
  0.2946    -0.31579    0.42943    0.086969   0.019169  -0.27242
 -0.31696    0.37327    0.61997    0.13889    0.17188    0.30363
 -1.2776     0.044423  -0.52736   -0.88536   -0.19428   -0.61947
 -0.10146   -0.26301   -0.061707   0.36627   -0.95223   -0.39346
 -0.69183   -1.0426     0.28855    0.63056  ]
(100,) (100,) (100,) [0.7592798]
```

Q1. Read the SemAxis paper and implement the SemAxis method described in sections 3.1.2 and 3.1.3. Given a set of word embeddings for positive terms $S^+ = \{v_1^+, \ldots v_n^+\}$ and embeddings for negative terms $S^- = \{v_1^-, \ldots v_n^-\}$ that define the endpoints of the axis, your output should be a single real-value score for an input word w with word representation v_w:

$$score(w)_{\mathbf{V}_{\text{axis}}} = \cos(v_w, \mathbf{V}_{\text{axis}})$$

Where:

$$\mathbf{V}^+ = \frac{1}{n}\sum_1^n v_i^+$$

$$\mathbf{V}^- = \frac{1}{m}\sum_1^m v_i^-$$

$$\mathbf{V}_{\text{axis}} = \mathbf{V}^+ - \mathbf{V}^-$$

In [40]:

```
def get_semaxis_score(vectors, positive_terms=None, negative_terms=None, ta

    # your code here
    positive_vecs=[]
    negative_vecs=[]
    for term in positive_terms:
        positive_vecs.append(glove[term])
    for term in negative_terms:
        negative_vecs.append(glove[term])
    pos_average=np.mean(positive_vecs, axis=0)
    neg_average=np.mean(negative_vecs, axis=0)
    v_axis = pos_average - neg_average
    # score should be a single real-value number (e.g., 0.342)
    score = glove.cosine_similarities(glove[target_word], [v_axis])
    return score[0]
```

In [41]:

```
# should be 0.342
get_semaxis_score(glove, positive_terms=["woman", "women"], negative_terms=
```

Out[41]:

```
0.3424988
```

Now let's score a set of target terms along that axis

In [42]:

```
def score_list_of_targets(vectors, positive_terms=None, negative_terms=None
    scores=[]
    for target in target_words:
        scores.append((get_semaxis_score(vectors, positive_terms, negative_

    for k,v in reversed(sorted(scores)):
        print("%.3f\t%s" % (k,v))
```

In [43]:
```
targets=["doctor", "nurse", "actor", "actress", "mechanic", "librarian", "a
```

In [44]:
```
score_list_of_targets(glove, positive_terms=["woman", "women"], negative_te
```

```
0.342   actress
0.294   nurse
0.219   librarian
0.106   doctor
0.024   actor
0.003   chef
-0.019  cook
-0.075  architect
-0.153  magician
-0.194  mechanic
```

Q2: Define your own concept axis by selecting a set of positive and negative terms and illustrate its utility by scoring a set of 10 target terms (as we did above).

In [45]:
```
positive_terms=['gentle']
negative_terms=['rough']
targets=['male','female','doctor','nurse','chef','engineer','teacher','pro

score_list_of_targets(glove, positive_terms=positive_terms, negative_terms=
```

```
0.147   professor
0.129   teacher
0.123   nurse
0.096   chef
0.053   female
0.038   male
0.010   doctor
-0.070  engineer
-0.161  cops
-0.254  police
```

In [46]:
```
positive_terms=['brown','dark']
negative_terms=['white','fair']
targets=['beautiful','ugly','pretty','handsome','attractive','unattractive
score_list_of_targets(glove, positive_terms=positive_terms, negative_terms=
```

```
0.175   handsome
0.153   ugly
0.104   beautiful
0.088   actor
0.048   pretty
0.047   actress
0.026   unattractive
-0.045  attractive
-0.109  undesirable
-0.140  model
-0.211  desirable
```

Q3: Let's assume now that you're able to score all words in a vocabulary along several conceptual dimensions (like the one you've defined) for a given set of word embeddings trained on a dataset. What could you do with that score? Brainstorm possible applications.

1. If the model is trained on a specialised dataset like tweets from supporters of 2 political candidates, it can be used to understand their stance on key issues.
2. Similarly, this can be applied to any scenario where we want to evaluate the stance of a community on an issue, even if it is as simple as product reviews, and how they span across different consumer groups.
3. It can help to understand societal perceptions like gender roles, beauty standards, etc. (As shown above)

In [ ]:

```

```