Prep

In [1]:
```
import collections
import numpy as np
import scipy
```

In [2]:
```
# I couldn't download nltk stopwords because python keeps giving me errors, so I manually down
loaded the txt file
# from nltk website and manually read it
wiki_full = open('wiki-text.txt',"r").readlines()[0].split()
stopwords = open('english.txt',"r").readlines()
stopwords = [s.replace('\n','') for s in stopwords]
```

In [122]:
```
wiki_sub = wiki_full[0:int(len(wiki_full)*(3/7))] # I use 3/7th of the wiki corpus
counts = dict(collections.Counter(wiki_sub))
counts = {key:value for (key,value) in counts.items() if value >= 500} # filter out infrequent
words
counts = {key:value for (key,value) in counts.items() if key not in stopwords} # filuter out s
topwords
len(counts)
```

Out[122]: 8124

Due to computing limitation my vocab size is around 8000.

(a)

In [135]:
```
vocabs = list(counts.keys())
vocabs_dict = dict(enumerate(vocabs))
vocabs_dict = dict((v,k) for k,v in vocabs_dict.items()) # this dictionary will act as indexes
for PMI-matrix.
```

In [ ]:
```
from collections import defaultdict
# create a dictionary that tracks list of all occurrences of a word in corpus
corpus_dict = defaultdict(list)
for index,word in enumerate(wiki_sub):
    if index % 1000000 ==0:
        print(index)
    if word in vocabs:
        corpus_dict[word].append(index)
```

In [ ]:
```
import scipy.sparse as sp
radius = list(range(-5,0)) + list(range(1,6)) # window of 5
total_words = len(wiki_sub)

# first create a matrix that counts co-occurences:
np_mat = sp.lil_matrix((len(counts),len(counts)))

for i in range(len(vocabs)):
    if i % 100 == 0:
        print(i)
    vocab_curr = vocabs[i]
    instances = corpus_dict[vocab_curr]
    temp = np.array(instances)
    context = np.zeros((len(instances),len(radius)))
    for j in range(len(radius)):
        context[:,j] = temp + radius[j]
    context = list(set(list(context.flatten('C'))))
    context = [int(k) for k in context if k>=0 and k<total_words]
    context_words = [wiki_sub[i] for i in context]
    context_words_dict = dict(collections.Counter(context_words))
    context_vocabs = list(set(context_words).intersection(set(vocabs)))
    if len(context_vocabs)>0:
        for word in context_vocabs:
            m = vocabs_dict[vocab_curr]
            n = vocabs_dict[word]
            num = context_words_dict[word]
            np_mat[(m,n)] += num
```

In [338]:
```
total_pairs = int(np_mat.sum())
marginal_pairs = np_mat.sum(axis=1)
outer = np.outer(marginal_pairs,marginal_pairs)
np_matd = np_mat.todense()
M = (np_matd + 1) * total_pairs
M = np.log(np.divide(M,outer))
```

(b)

In [356]:
```
sM = scipy.sparse.csr_matrix(M)
from scipy.sparse.linalg import svds, eigs
U,s,V = svds(sM,k=50)
```

(c)

In [371]:
```
sigma = np.sqrt(np.diag(s))
W = np.matmul(U,sigma)
W.shape
```

Out[371]: (8124, 50)

(d)

In [385]:
```
index_dict = dict(enumerate(vocabs))
vocabs_dict = dict((v,k) for k,v in index_dict.items())
```

In [386]:
```
from scipy.spatial import distance
dist = distance.cdist(W, W, 'euclidean')
```

In [387]:
```
word_to_check = ['physics','republican','einstein','algebra','fish']
word_to_check_indexes = []
for word in word_to_check:
    word_to_check_indexes.append(vocabs_dict[word])
```

In [388]:
```
word_to_check_indexes
```

Out[388]: [3669, 1174, 7181, 4383, 4485]

In [412]:
```
def get_top5_words(index):
    word_dist = dist[index,:]
    ind = np.argpartition(word_dist,5)[0:5]
    res = []
    for i in range(5):
        res.append(index_dict[ind[i]])
    return(res)
```

In [413]:
```
top5s = []
for i in word_to_check_indexes:
    res = get_top5_words(i)
    top5s.append(res)
print(top5s)
```

```
[['mathematics', 'quantum', 'mechanics', 'physics', 'chemistry'], ['republican', 'democrats',
'presidential', 'candidate', 'electoral'], ['maxwell', 'newton', 'relativity', 'einstein', 'p
aradox'], ['algebraic', 'algebra', 'topology', 'finite', 'equations'], ['fruit', 'fish', 'mea
t', 'trees', 'milk']]
```

The 5 closest words in embedding space all make sense for each of the five words.

(e)

I will explore the following analogies:

```
france:paris = england :?
republican:democrat = conservative:?
man:woman = male:?
```

In [423]:
```
group1 = ['paris','france','england']
group2 = ['democrat','republican','conservative']
group3 = ['woman','man','male']
```

In [460]:
```
def get_analogies(group):
    indexes = []
    for word in group:
        indexes.append(vocabs_dict[word])
    v1 = W[indexes[0],:]
    v2 = W[indexes[1],:]
    v3 = W[indexes[2],:]
    v = np.add(np.subtract(v1,v2),v3).reshape(1,50)
    dist_v = distance.cdist(W, v, 'euclidean').flatten()
    top5_ind = np.argpartition(dist_v,5)[0:5]
    res = []
    for ind in top5_ind:
        res.append(index_dict[ind])
    return(res)
```

In [461]:
```
res1 = get_analogies(group1)
res1
```

Out[461]: ['hall', 'london', 'castle', 'boston', 'oxford']

In [462]:
```
res2 = get_analogies(group2)
res2
```

Out[462]: ['conservatives', 'evangelical', 'liberals', 'criticisms', 'reject']

In [463]:
```
res3 = get_analogies(group3)
res3
```

Out[463]: ['families', 'sex', 'female', 'male', 'couples']

Our desired outputs are: england to london, conservative to liberal, and male to female, and from our embedding we can get all these outputs from the top 5 nearest embedding vector to the linear combination of the rest of the embedding vectors in the analogies.