2. Word Embedding Experiments (40 points)

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
from scipy.sparse import csr_matrix
from scipy.sparse import linalg
from math import log
import pickle
```

NOTE: you can reload the W matrix and the vocab from down below in the line that uses pickle if you dont want to recalculate the matrices.

The following experiments should be done with the Wikipedia corpus here: /project2/cmsc25025/wikipedia/wiki-text.txt The number of unique words in the Wikipedia corpus is too large for our purposes. Before proceeding, you should come up with a smaller vocabulary V that you will use for the remainder of the word embedding experiments. You can filter all of the unique words in the Wikipedia corpus in a number of ways. Here are some examples:

• Remove words that appear less than n times (e.g. try n = 500). You may use any existing python packages to compute word counts, for example nltk.FreqDist or collections.Counter. • Remove all words that appear in the stopwords list of nltk package.

```
from nltk.corpus import stopwords
import nltk
nltk.download(’stopwords’)
stop_words = set(stopwords.words(’english’))
```

You should aim to have roughly 15,000 words in your vocabulary. In what follows, you may simply ignore words that do not appear in your final filtered vocabulary V . Whatever code you run try it first on a small section of the wiki data. The final run may take a while.

In [2]:
```
sw = {'ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'there', '
```

In [9]:
```
words = []
with open('/project2/cmsc25025/wikipedia/wiki-text.txt') as f:
    for ws in f:
        words = ws.split()
```

In [10]:
```
word_counter = Counter(words)
```

```
In [52]: required_n = 500
         vocab = list(filter(lambda x: x[1] > required_n and x[0] not in sw, word_cou
         vocab = set([x[0] for x in vocab])

         words = list(filter(lambda x: x in vocab, words))
```

(a) Using the Wikipedia data with a symmetric context window size of 5, compute the PMI matrix $M \in R^{N(V) \times N(V)}$ whose (i,j)-th entry is the PMI of (w_i, w_j). To avoid degeneracy, add 1 to the cooccurence statistics $N^p(w_i, w_j)$ so that

$$M_{ij} = \log\left(\frac{\left(N^p\left(w_i, w_j\right) + 1\right) \cdot N\left(S^p\right)}{N^p\left(w_i\right) \cdot N^p\left(w_j\right)} \right)$$

where S^p is the set of all word-context pairs observed in the Wikipedia corpus. Note that $N^p(w_i, w_j)$ is simply the number of times w_i and w_j cooccur within 5 words of each other across the entire corpus.

```
In [53]: def createContexts(words):
             counter_wc = Counter()
             counter_w = Counter()
             ns = 0
             for (i,w) in enumerate(words):
                 for j in list(range(max(0, i-5),i)) + list(range(i+1,min(i+6,len(wor
                     counter_wc[(w,words[j])] += 1
                     counter_w[w] += 1
                     ns += 1
             return counter_wc, counter_w, ns

         nwc, nw, ns = createContexts(words)
```

```
In [54]: def getMij(wi, wj):
             return log(((nwc[(wi,wj)] + 1) * ns)/(nw[wi]*nw[wj]))

         M = np.array([[getMij(wi,wj) for wj in vocab] for wi in vocab])
```

(b) Now we will factorize the matrix M to obtain word embeddings. Take the k-SVD of M with k=50

$$M = U\Sigma V^T = U\Sigma^{\frac{1}{2}}\Sigma^{\frac{1}{2}}V^T$$

where $U \in \mathbb{R}^{|V| \times 50}, \Sigma \in \mathbb{R}^{50 \times 50}$, and $V^T \in \mathbb{R}^{50 \times |V|}$ You may use the scipy package to compute the k-SVD:

```
In [56]: U, sigma, V = linalg.svds(csr_matrix(M), k=50)
```

(c) Let the rows of $W = U\Sigma^{1/2}$ be the learned PMI word embeddings. You will use these embeddings in several tasks below. We recommend that you serialize these embeddings to disk for later use (e.g., by using the cPickle package).

In [57]:
```
W = U.dot(np.diag(sigma**0.5))
```

In [58]:
```
pickle.dump(W, open( "W_PMI.p", "wb" ))
pickle.dump(list(vocab), open( "vocab_PMI.p", "wb" ))
```

Reload from here if you don't want to recalculate the matrix.

In [3]:
```
W = pickle.load( open( "W_PMI.p", "rb" ) )
vocab = pickle.load( open( "vocab_PMI.p", "rb" ))
```

(d) For each of the following words, find the 5 closest words in the embedding space:

```
physics, republican, einstein, algebra, fish.
```

In [4]:
```
def getClosest(W, vocab, w_key, num = 6):
    closest = [(vocab[i], np.linalg.norm(w_key - w)) for (i,w) in enumerate(
    closest.sort(key = lambda x: x[1])

    closest = [x[0] for x in closest[:num]]
    return closest
```

In [5]:
```
key_words = ["physics", 'republican', 'einstein', 'algebra', 'fish']
close_words = []
for key in key_words:
    w_key = W[vocab.index(key)]
    close_words.append(getClosest(W, vocab, w_key)[1:])
for i,key in enumerate(key_words):
    print("The 5 closest words to", key, "are", close_words[i])
```

```
The 5 closest words to physics are ['mechanics', 'quantum', 'chemistry',
'theoretical', 'mathematics']
The 5 closest words to republican are ['senator', 'democrat', 'democrat
s', 'candidate', 'presidential']
The 5 closest words to einstein are ['relativity', 'physicists', 'parado
x', 'maxwell', 'experiment']
The 5 closest words to algebra are ['algebraic', 'finite', 'theorem', 'to
pology', 'calculus']
The 5 closest words to fish are ['fruit', 'eggs', 'eat', 'seeds', 'meat']
```

These all seem like similar words and thus the model performed extremely well in this situation. I was amazed by how amazing this performed.

(e) A surprising consequence of some word embedding methods is the resulting linear substructure. This structure can be used to solve analogies like

```
france :  paris ::  england :  ?
```

by computing the nearest embedding vector to v where v is

```
v = v_{paris} - v_{france} + v_{england}
```

Define 3 analogies X:Y=Z:W. and report the top 5 words you get.

In [63]:
```
def getAnalogyV(x, y, z):
    wx = W[vocab.index(x)]
    wy = W[vocab.index(y)]
    wz = W[vocab.index(z)]
    return wx - wy + wz

analogy_words = [["republican", "democrat", "conservative"], ["wet", "dry",
closest_analogy = []
for i in range(len(analogy_words)):
    v = getAnalogyV(analogy_words[i][0], analogy_words[i][1], analogy_words
    closest_analogy.append(getClosest(W, vocab, v, num = 5))
for i,analogy in enumerate(analogy_words):
    print(analogy[0],":",analogy[1],"::",analogy[2],":",closest_analogy[i])
```

```
republican : democrat :: conservative : ['conservative', 'liberal', 'oppo
sition', 'leadership', 'leaders']
wet : dry :: hot : ['cool', 'smoke', 'flame', 'burn', 'dust']
taste : sweet :: smell : ['sensation', 'unpleasant', 'treating', 'smell',
'defects']
```

The analogies did not perform as well as the similar words, but the words returned are still defintely relevant to each other, especially the first few. It is hard to have this model return a strong analogy given the fact that the model only looks at similar relevant words and not definitions or any other sort of relevant metadata. Many words have multiple uses and meanings, which makes it difficult to judge the word.

Present Slides Themes Help