Problem Set #5

ECE 4424 / CS 4824 - Machine learning
Virginia Tech

November 21, 2020

- Feel free to collaborate with other classmates in doing the homework. Please indicate your collaborators with their student ID. You should, however, write down your solution yourself. Please try to keep the answers brief and clear.
- Whenever you need clarification, please post the related questions on Piazza under the corresponding homework folder.
- Total: 100 points
- **Due date: 12/5/2020, 11:59PM ET**
- Late submission: each student will have a total of **four** free late (calendar) days to use for homeworks. **Note: this is the total number of late days accumulated through all the homeworks so far. It's NOT reset after each homework!** Once these late days are exhausted, any assignments turned in late will be penalized 20% per late day. However, no assignment will be accepted more than three days after its due date. Each 24 hours or part thereof that a homework is late uses up one full late day.

1 K-Means algorithm (30 pts)

Given the dataset which contains points $x_1 = (1, 2)$, $x_2 = (2, 2)$, $x_3 = (2, 1)$, $x_4 = (-1, 5)$, $x_5 = (-2, -1)$, $x_6 = (-1, -1)$. Suppose we want to have 2 clusters. Answer the following questions:

a) (15 pts) Initialize the clusters cluster by $\mu_1 = x_1$ and $\mu_2 = x_4$, run the K-means clustering algorithm and report the final clusters (in terms of the points in each cluster and the cluster centers). Use L1 distance as the distance between points which is given by

$$d(x, y) = \sum_{j=1}^{d} |x^{(j)} - y^{(j)}|$$

where $x^{(j)}$ is the $j-$th entry of $x \in \mathbb{R}^d$.

a.) $\mu_1 = x_1 = (1,2)$ $\mu_2 = x_4 = (-1,5)$

$x_1 = (1,2)$ $x_2 = (2,2)$ $x_3 = (2,1)$

$x_4 = (-1,5)$ $x_5 = (-2,-1)$ $x_6 = (-1,-1)$

$\mu_1 - x_2 : |1-2| + |2-2| = 1$

$\mu_1 - x_3 : |1-2| + |2-1| = 2$

$\mu_1 - x_5 : |1-(-2)| + |2-(-1)| = 6$

$\mu_1 - x_6 : |1-(-1)| + |2-(-1)| = 5$

$\mu_2 - x_2 : |(-1)-2| + |5-2| = 6$

$\mu_2 - x_3 : |(-1)-2| + |5-1| = 7$

$\mu_2 - x_5 : |(-1)-(-2)| + |5-1| = 5$

$\mu_2 - x_6 : |(-1)-(-1)| + |5-(-1)| = 6$

Cluster #1: x_1, x_2, x_3, x_6

Average $= \left(\frac{(1+2+2+(-1))}{4}, \frac{(2+2+1+(-1))}{4}\right)$

new $\mu_1 = (1,1)$

Cluster: x_4, x_5

Average $= \left(\frac{(-1+-2)}{2}, \frac{(5+-1)}{2}\right)$

new $\mu_2 = \left(-\frac{3}{2}, 2\right)$

new μ_1 & μ_2

$\mu_1 - x_1 : |1-1| + |1-2| = 1$

$\mu_1 - x_2 : |1-2| + |1-2| = 2$

$\mu_1 - x_3 : |1-2| + |1-1| = 1$

$\mu_1 - x_4 : |1-(-1)| + |1-5| = 6$

$\mu_1 - x_5 : |1-(-2)| + |1-(-1)| = 5$

$\mu_1 - x_6 : |1-(-1)| + |1-(-1)| = 4$

$\mu_2 - x_1 : |(-\frac{3}{2})-1| + |2-2| = 2.5$

$\mu_2 - x_2 : |(-\frac{3}{2})-2| + |2-2| = 3.5$

$\mu_2 - x_3 : |(-\frac{3}{2})-2| + |2-1| = 4.5$

$\mu_2 - x_4 : |(-\frac{3}{2})-(-1)| + |2-5| = 3.5$

$\mu_2 - x_5 : |(-\frac{3}{2})-(-2)| + |2-(-1)| = 3.5$

$\mu_2 - x_6 : |(-\frac{3}{2})-(-1)| + |2-(-1)| = 3.5$

Cluster: x_1, x_2, x_3

$\mu_1 = (1,1)$

Average $= \left(\frac{(1+2+2)}{3}, \frac{(2+2+1)}{3}\right)$

new $\mu_1 = \left(\frac{5}{3}, \frac{5}{3}\right)$

Cluster: x_4, x_5, x_6

$\mu_2 = (-\frac{3}{2}, 2)$

Average $= \left(\frac{(-1+-2+-1)}{3}, \frac{(5+-1+-1)}{3}\right)$

new $\mu_2 = \left(-\frac{4}{3}, 1\right)$

new m_1 & m_2

$m_1 - x_1 : \left|\frac{5}{3} - 1\right| + \left|\frac{5}{3} - 2\right| = 1$

$m_1 - x_2 : \left|\frac{5}{3} - 2\right| + \left|\frac{5}{3} - 2\right| = 0.66$

$m_1 - x_3 : \left|\frac{5}{3} - 2\right| + \left|\frac{5}{3} - 1\right| = 1$

$m_1 - x_4 : \left|\frac{5}{3} - (-1)\right| + \left|\frac{5}{3} - 5\right| = 6$

$m_1 - x_5 : \left|\frac{5}{3} - (-2)\right| + \left|\frac{5}{3} - (-1)\right| = 6.33$

$m_1 - x_6 : \left|\frac{5}{3} - (-1)\right| + \left|\frac{5}{3} - (-1)\right| = 5.33$

Final Cluster : x_1, x_2, x_3

Cluster center : $(\frac{5}{3}, \frac{5}{3})$

$m_2 - x_1 : \left|(-\frac{4}{3}) - 1\right| + \left|1 - 2\right| = 3.33$

$m_2 - x_2 : \left|(-\frac{4}{3}) - 2\right| + \left|1 - 2\right| = 4.33$

$m_2 - x_3 : \left|(-\frac{4}{3}) - 2\right| + \left|1 - 1\right| = 3.33$

$m_2 - x_4 : \left|(-\frac{4}{3}) - (-1)\right| + \left|1 - 5\right| = 4.33$

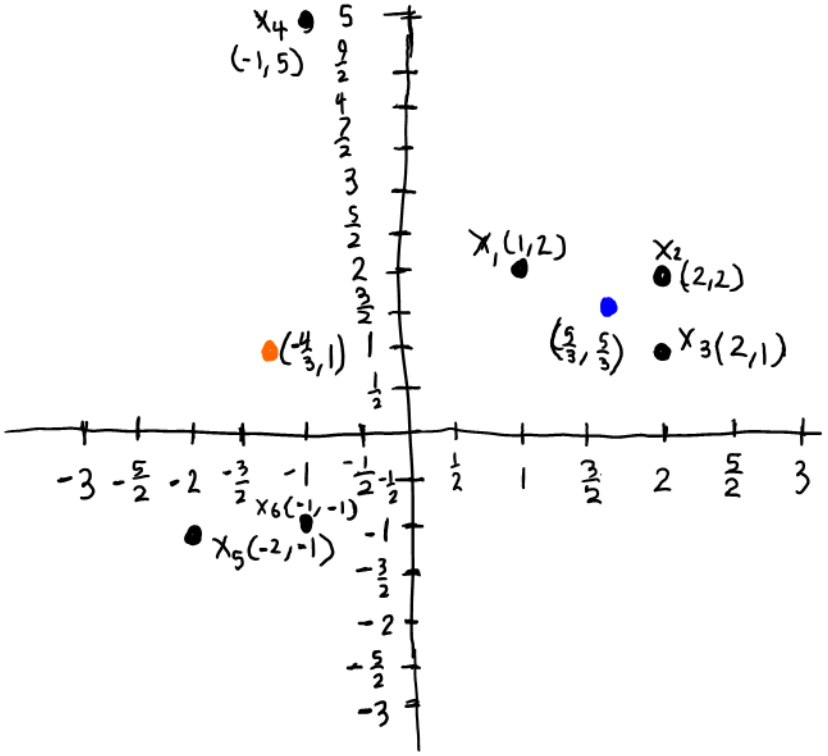
$m_2 - x_5 : \left|(-\frac{4}{3}) - (-2)\right| + \left|1 - (-1)\right| = 2.67$

$m_2 - x_6 : \left|(-\frac{4}{3}) - (-1)\right| + \left|1 - (-1)\right| = 2.33$

Final Cluster : x_4, x_5, x_6

Cluster center : $(-\frac{4}{3}, 1)$

b.)



Center 1

Center 2

C.)

$\mu_1 = X_1 (1,2) \quad \mu_2 = X_5 (-2,-1)$

$\mu_1 - x_2 : |1-2| + |2-2| = 1$

$\mu_1 - x_3 : |1-2| + |2-1| = 2$

$\mu_1 - x_4 : |1-(-1)| + |2-5| = 5$

$\mu_1 - x_6 : |1-(-1)| + |2-(-1)| = 5$

$\mu_2 - x_2 : |(-2)-2| + |(-1)-2| = 7$

$\mu_2 - x_3 : |(-2)-2| + |(-1)-1| = 6$

$\mu_2 - x_4 : |(-2)-(-1)| + |(-1)-5| = 7$

$\mu_2 - x_6 : |(-2)-(-1)| + |(-1)-(-1)| = 1$

$X_1 = (1,2) \quad X_2 = (2,2) \quad X_3 = (2,1)$

$X_4 = (-1,5) \quad X_5 = (-2,-1) \quad X_6 = (-1,-1)$

Clusters: X_1, X_2, X_3, X_4

Average: $\left(\frac{(1+2+2+-1)}{4}, \frac{(2+2+1+5)}{4}\right)$

new $\mu_1 = \left(1, \frac{5}{2}\right)$

Clusters: X_5, X_6

Average: $\left(\frac{(-2+-1)}{2}, \frac{(-1+-1)}{2}\right)$

new $\mu_2 = \left(-\frac{3}{2}, -1\right)$

new $\mu_1 \left(1, \frac{5}{2}\right)$ & $\mu_2 \left(-\frac{3}{2}, -1\right)$

$\mu_1 - X_1 : |1-1| + |\frac{5}{2}-2| = 0.5$

$\mu_1 - X_2 : |1-2| + |\frac{5}{2}-2| = 1.5$

$\mu_1 - X_3 : |1-2| + |\frac{5}{2}-1| = 2.5$

$\mu_1 - X_4 : |1-(-1)| + |\frac{5}{2}-5| = 4.5$

$\mu_1 - X_5 : |1-(-2)| + |\frac{5}{2}-(-1)| = 6.5$

$\mu_1 - X_6 : |1-(-1)| + |\frac{5}{2}-(-1)| = 5.5$

$\mu_2 - X_1 : |(-\frac{3}{2})-1| + |(-1)-2| = 5.5$

$\mu_2 - X_2 : |(-\frac{3}{2})-2| + |(-1)-2| = 6.5$

$\mu_2 - X_3 : |(-\frac{3}{2})-2| + |(-1)-1| = 5.5$

$\mu_2 - X_4 : |(-\frac{3}{2})-(-1)| + |(-1)-5| = 6.5$

$\mu_2 - X_5 : |(-\frac{3}{2})-(-2)| + |(-1)-(-1)| = 0.5$

$\mu_2 - X_6 : |(-\frac{3}{2})-(-1)| + |(-1)-(-1)| = 0.5$

Final Cluster: X_1, X_2, X_3, X_4

Cluster Center: $\left(1, \frac{5}{2}\right)$

Final Cluster: X_5, X_6

Cluster Center: $\left(-\frac{3}{2}, -1\right)$

By choosing X_1 and X_5 as new initialization points, we have created a different clustering

b) (5 pts) Draw the points on a 2-D grid and check if the clusters make sense.

c) (10 pts) Can you find an initialization that results in a different clustering?

2 Programming assignment (70 pts)

For the following programming assignment, please download the datasets and iPython notebooks from Canvas and submit the following:

- Completed and ready-to-run iPython notebooks. Note: we will inspect the code and run your notebook if needed. If we cannot run any section of your notebook, you will not receive any points for the task related to that section.
- Responses (texts, codes, and/or figures) to the following problems/tasks

In this programming assignment, we will experiment with distributed representations of words. We'll also see how such an embedding can be constructed by applying principal component analysis to a suitably transformed matrix of word co-occurrence probabilities.

Task P1 (5 pts): Complete the following code to get a list of words and their counts. Report how many times does the word "evidence" and "investigation" appears in the corpus.

Task P2 (10 pts): Decide on the vocabulary. There are two potentially distinct vocabularies: the words for which we will obtain embeddings ('vocab_words') and the words we will consider when looking at context information ('context_words'). We will take the former to be all words that occur at least 20 times, and the latter to be all words that occur at least 100 times. We will stick to these choices for this assignment, but feel free to play around with them and find something better. Also, report the sizes of these two word lists.

Task P3 (10 pts): Get co-occurrence counts. These are defined as follows, for a small constant 'window_size=2':

- Let 'w0' be any word in 'vocab_words' and 'w' any word in 'context_words'.
- Each time 'w0' occurs in the corpus, look at the window of 'window_size' words before and after it. If 'w' appears in this window, we say it appears in the context of (this particular occurrence of) 'w0'.
- Define 'counts[w0][w]' as the total number of times 'w' occurs in the context of 'w0'.

Complete the function 'get_counts', which computes the 'counts' array, and returns it as a dictionary (of dictionaries). Find how many times the word "fact" appears in the context of "evidence" with window_size=2.

Task P4 (10 pts): Define 'probs[w0][]' to be the distribution over the context of 'w0', that is:

probs[w0][w] = counts[w0][w] / (sum of all counts[w0][])

Finish the function 'get_co_occurrence_dictionary' that computes 'probs'. Find the probability that the word "fact" appears in the context of "evidence".

Task P5 (10 pts): Based on the various pieces of information above, we compute the pointwise mutual information matrix (PMI):

$$\text{PMI}[i, j] = \max\left(0, \log \frac{\text{probs[i-th vocab word][j-th context word]}}{\text{context\_frequency[j-th context word]}}\right)$$

Complete the code to compute PMI for every word i and context word j. Report the output of the code.

Task P6 (10 pts): Implement the following function that finds the nearest neighbor of a given word in the embedded space. Note down the answers to the following queries.

Task P7 (15 pts): Implement the function that aims to solve the analogy problem:

A is to B as C is to ?

For example, A=King, B=Queen, C=man, and the answer for ? should be ideally woman (you will see that this may not be the case using the distributed representation).

Finds the K-nearest neighbor of a given word in the embedded space. Note: instead of outputting only the nearest neighbor, you should find the K=10 nearest neighbors and see whether there is one in the list that makes sense. You should also exclude the words C in the output list.

Also report another set A, B, C and the corresponding answer output by your problem. See if it makes sense to you.

Answers to Programming Attached Below

Distributed Representation for Word Embeddings

In this programming assignment, we will experiment with distributed representations of words. We'll also see how such an embedding can be constructed by applying principal component analysis to a suitably transformed matrix of word co-occurrence probabilities. For computational reasons, we'll use the moderately sized **Brown corpus of present-day American English** for this.

1. Accessing the Brown corpus

The *Brown corpus* is available as part of the Python Natural Language Toolkit (`nltk`).

In [1]:
```
import numpy as np
import pickle
import nltk
nltk.download('brown')
nltk.download('stopwords')
from nltk.corpus import brown, stopwords
from scipy.cluster.vq import kmeans2
from sklearn.decomposition import PCA
```

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

The corpus consists of 500 samples of text drawn from a wide range of sources. When these are concatenated, they form a very long stream of over a million words, which is available as `brown.words()` . Let's look at the first 50 words.

In [2]:
```
for i in range(20):
    print (brown.words()[i],)
```

```
The
Fulton
County
Grand
Jury
said
Friday
an
investigation
of
Atlanta's
recent
primary
election
produced
``
no
evidence
''
that
```

Before doing anything else, let's remove stopwords and punctuation and make everything lowercase. The resulting sequence will be stored in `my_word_stream` .

In [3]:
```
my_stopwords = set(stopwords.words('english'))
word_stream = [str(w).lower() for w in brown.words() if w.lower() not in my_st
opwords]
my_word_stream = [w for w in word_stream if (len(w) > 1 and w.isalnum())]
```

Here are the initial 20 words in `my_word_stream` .

In [4]:
```
my_word_stream[:20]
```

Out[4]:
```
['fulton',
 'county',
 'grand',
 'jury',
 'said',
 'friday',
 'investigation',
 'recent',
 'primary',
 'election',
 'produced',
 'evidence',
 'irregularities',
 'took',
 'place',
 'jury',
 'said',
 'presentments',
 'city',
 'executive']
```

2. Computing co-occurrence probabilities

Task P1: Complete the following code to get a list of words and their counts. Report how many times does the word "evidence" and "investigation" appears in the corpus.

In [5]:
```
N = len(my_word_stream)
# print(N)
words = []
totals = {}

## STUDENT: Your code here
# words: a python list of unique words in the document my_word_stream as the v
ocabulary
# totals: a python dictionary, where each word is a key, and the corresponding
value
#         is the number of times this word appears in the document my_word_str
eam

words = my_word_stream

for word in words:
    if word in totals:
        totals[word] += 1
    else:
        totals[word] = 1


## STUDENT CODE ENDS
```

In [6]:
```
## STUDENT: Report how many times does the word "evidence" and "investigation"
appears in the corpus.
print('Word "',words[11],'" appears ',totals[words[11]], ' times')
print('Word "',words[6],'" appears ',totals[words[6]], ' times')
```

```
Word " evidence " appears  204  times
Word " investigation " appears  51  times
```

Task P2: Decide on the vocabulary. There are two potentially distinct vocabularies: the words for which we will obtain embeddings (`vocab_words`) and the words we will consider when looking at context information (`context_words`). We will take the former to be all words that occur at least 20 times, and the latter to be all words that occur at least 100 times. We will stick to these choices for this assignment, but feel free to play around with them and find something better.

How large are these two word lists? Note down these numbers.

In [7]:
```
## STUDENT: Your code here

vocab_words = [] # a list of words whose occurances (totals) are > 19
context_words = [] # a list of words whose occurances (totals) are > 99

wordList = totals.items()
for word in wordList:
    if word[1] > 99:
        context_words.append(word[0])
    if word[1] > 19:
        vocab_words.append(word[0])

## STUDENT CODE ENDS
print ('Number of vocabulary words ',len(vocab_words), ';')
print ('Number of context words ',len(context_words), ';' )
#print(context_words)
```

```
Number of vocabulary words  4720 ;
Number of context words  918 ;
```

Task P3: Get co-occurrence counts. These are defined as follows, for a small constant `window_size=2` .

- Let `w0` be any word in `vocab_words` and `w` any word in `context_words` .
- Each time `w0` occurs in the corpus, look at the window of `window_size` words before and after it. If `w` appears in this window, we say it appears in the context of (this particular occurrence of) `w0` .
- Define `counts[w0][w]` as the total number of times `w` occurs in the context of `w0` .

Complete the function `get_counts` , which computes the `counts` array and returns it as a dictionary (of dictionaries). Find how many times the word "fact" appears in the context of ”evidence" with window_size=2.

In [8]:
```
def get_counts(window_size=2):
    ## Input:
    #  window_size: for each word w0, its context includes window_size words b
efore and after it.
    #  For instance, if window_size = 2, it means we look at w1 w2 w0 w3 w4, w
here  w1, w2, w3, w4 are
    #  context woreds
    ## Output:
    #  counts: a python dictionary (of dictionaries) where counts[w0][w] indic
ate the number of times the word w appears
    #  in the context of w0 (Note: counts[w0] is also a python dictionary)

    counts = {}
    for w0 in vocab_words:
        counts[w0] = {}

    ## STUDENT: Your code here

    #set each vocab word to have counter for all possible context words
    for w0 in vocab_words:
        for word in context_words:
            counts[w0][word]= 0

    vocab_list = list(counts.keys())
    words = my_word_stream
    index_list = []
    window_list = []
    vocab_index = 0
    indexMin = 0
    indexMax = 0
    word_of_interest = ""

    #iterate through the vocab words
    for i in range(len(vocab_list)):

        #get the current vocab_word
        word_of_interest = vocab_list[i]

        #find the index of every occurence of the current vocab word in the co
rpus
        index_list = [i for i,word in enumerate(words) if word == word_of_inte
rest]

        #print(index_list)
        #go to every occurence of the vocab word in the corpus and check the w
indow size
        for i in index_list:

            indexMin = i - window_size
            indexMax = i + window_size + 1

            if indexMin < 0:
                indexMin = 0

            if indexMax > len(words) - 1:
                indexMax = len(words)
```

```
            window_list = words[indexMin:indexMax]
            window_list.remove(word_of_interest)
            #print(window_list)

            for word in context_words:
                if word in window_list:
                    counts[word_of_interest][word] += 1

    ## End of codes
    return counts
```

In [9]:
```
## STUDENT: Report how many times the word "fact" appears in the context of ”evidence".
counts = get_counts(window_size=2)
print (counts['evidence']['fact'])
```

```
4
```

Define `probs[w0][]` to be the distribution over the context of `w0`, that is:

- `probs[w0][w] = counts[w0][w] / (sum of all counts[w0][])`

Task P4: Finish the function `get_co_occurrence_dictionary` that computes `probs`. Find the probability that the word "fact" appears in the context of ”evidence".

In [10]:
```
def get_co_occurrence_dictionary(counts):
    ## Input:
    #  counts: a python dictionary (of dictionaries) where counts[w0][w] indicate the number of times the word w appears
    #  in the context of w0 (Note: counts[w0] is also a python dictionary)
    ## Output:
    #  probs: a python dictionary (of dictionaries) where probs[w0][w] indicate the probability that word w appears
    #  in the context of word w0

    probs = {}

    ## STUDENT: Your code here
    #initialize probability dictionary to have each vocab word have a probability of every context word be initially zero
    for w0 in vocab_words:
        probs[w0] = {}
        for word in context_words:
            probs[w0][word]= 0

    for w0 in list(probs.keys()):
        for w in list(probs[w0].keys()):
            probs[w0][w] = counts[w0][w]/sum( list( counts[w0].values() ) )

    ## End of codes
    return probs
```

In [11]:
```
## STUDENT: Report how many times the word "fact" appears in the context of "evidence".
probs = get_co_occurrence_dictionary(counts)
print (probs['evidence']['fact'])
```

```
0.010723860589812333
```

The final piece of information we need is the frequency of different context words. The function below, `get_context_word_distribution`, takes `counts` as input and returns (again, in dictionary form) the array:

- `context_frequency[w]` = sum of all `counts[][w]` / sum of all `counts[][]`

In [12]:
```
def get_context_word_distribution(counts):
    counts_context = {}
    sum_context = 0
    context_frequency = {}
    for w in context_words:
        counts_context[w] = 0
    for w0 in counts.keys():
        for w in counts[w0].keys():
            counts_context[w] = counts_context[w] + counts[w0][w]
            sum_context = sum_context + counts[w0][w]
    for w in context_words:
        context_frequency[w] = float(counts_context[w])/float(sum_context)
    return context_frequency
```

3. The embedding

Task P5: Based on the various pieces of information above, we compute the **pointwise mutual information matrix**:

- `PMI[i,j] = MAX(0, log probs[ith vocab word][jth context word] - log context_frequency[jth context word])`

Complete the code to compute PMI for every word i and context word j. Report the output of the code.

In [13]:
```
print ("Computing counts and distributions")
counts = get_counts(2)
probs = get_co_occurrence_dictionary(counts)
context_frequency = get_context_word_distribution(counts)
#
print ("Computing pointwise mutual information")
n_vocab = len(vocab_words)
n_context = len(context_words)
pmi = np.zeros((n_vocab, n_context))
for i in range(0, n_vocab):
    w0 = vocab_words[i]
    for w in probs[w0].keys():
        j = context_words.index(w)
        ## STUDENT: Your code here
        pmi[i,j] = max( 0, np.log(probs[w0][context_words[j]] ) - np.log(conte
xt_frequency[context_words[j]]) )
        ## Student end of code
```

```
Computing counts and distributions
Computing pointwise mutual information

<ipython-input-13-c9b5bb3dc9ab>:15: RuntimeWarning: divide by zero encountere
d in log
  pmi[i,j] = max( 0, np.log(probs[w0][context_words[j]] ) - np.log(context_fr
equency[context_words[j]]) )
```

In [14]:
```
# STUDENT: report the following number
print(pmi[vocab_words.index('evidence'),context_words.index('fact')])
```

```
1.6800791816244818
```

The embedding of any word can then be taken as the corresponding row of this matrix. However, to reduce the dimension, we will apply principal component analysis (PCA).

See this nice tutorial on PCA: https://www.youtube.com/watch?v=fkf4IBRSeEc (https://www.youtube.com/watch?v=fkf4IBRSeEc)

Now reduce the dimension of the PMI vectors using principal component analysis. Here we bring it down to 100 dimensions, and then normalize the vectors to unit length.

In [15]:
```
pca = PCA(n_components=100)
vecs = pca.fit_transform(pmi)
for i in range(0,n_vocab):
    vecs[i] = vecs[i]/np.linalg.norm(vecs[i])
```

It is useful to save this embedding so that it doesn't need to be computed every time.

In [16]:
```
fd = open("embedding.pickle", "wb")
pickle.dump(vocab_words, fd)
pickle.dump(context_words, fd)
pickle.dump(vecs, fd)
fd.close()
```

4. Experimenting with the embedding

We can get some insight into the embedding by looking at some intersting use cases.

Task P6: Implement the following function that finds the nearest neighbor of a given word in the embedded space. Note down the answers to the following queries.

In [17]:
```
def word_NN(w,vecs,vocab_words,context_words):
    ## Input:
    #  w: word w
    #  vecs: the embedding of words, as computed above
    #  vocab_words: vocabulary words, as computed in Task P2
    #  context_words: context words, as computed in Task P2
    ## Output:
    #  the nearest neighbor (word) to word w
    if not(w in vocab_words):
        print ("Unknown word")
        return

    ## Student: your code here
    vect = vecs[vocab_words.index(w)]
    neighbor = 0
    curr_dist = np.linalg.norm(vect - vecs[0])
    for i in range(1, len(vocab_words)):
        dist = np.linalg.norm(vect - vecs[i])
        if(dist < curr_dist) and (dist > 0.0):
            neighbor = i
            curr_dist = dist

    return vocab_words[neighbor]
    ## Student: code ends
```

In [18]:
```
word_NN('world',vecs,vocab_words,context_words)
```

Out[18]: 'nations'

In [19]:
```
word_NN('learning',vecs,vocab_words,context_words)
```

Out[19]: 'freedom'

In [20]:
```
word_NN('technology',vecs,vocab_words,context_words)
```

Out[20]: 'studies'

In [21]:
```
word_NN('man',vecs,vocab_words,context_words)
```

Out[21]:
```
'woman'
```

Task P7: Implement the function that aims to solve the analogy problem: A is to B as C is to ? For example, A=King, B=Queen, C=man, and the answer for ? should be ideally woman (you will see that this may not be the case using the distributed representation).

Finds the K-nearest neighbor of a given word in the embedded space. Note: instead of outputing only the nearest neighbor, you should find the K=10 nearest neighbors and see whether there is one in the list that makes sense. You should also exclude the words C in the output list.

Also report another set A, B, C and the corresponding answer output by your problem. See if it makes sense to you.

In [59]:
```
def find_analogy(A,B,C,vecs,vocab_words,context_words):
    ## Input:
    #  A, B, C: words A, B, C
    #  vecs: the embedding of words, as computed above
    #  vocab_words: vocabulary words, as computed in Task P2
    #  context_words: context words, as computed in Task P2
    ## Output:
    #  the word that solves the analogy problem
    ## STUDENT: Your code
    import operator
    query_word = vecs[vocab_words.index(C)] + vecs[vocab_words.index(B)] - vec
s[vocab_words.index(A)]
    distances = []
    for index in range(1, len(vocab_words)):
        dist = np.linalg.norm(query_word - vecs[index])

        if(vocab_words[index] != A) and (vocab_words[index] != B) and (vocab_w
ords[index] != C):
            distances.append([index, dist, vocab_words[index]])

    distances.sort(key=operator.itemgetter(1))

    neighbors = []

    for i in range(10):
        neighbors.append(distances[i][0])

    output_vals = []
    for i in range(len(neighbors)):
        output_vals.append(vocab_words[neighbors[i]])

    return output_vals
    ## STUDENT: your code ends
```

In [60]:
```
find_analogy('king','queen','man',vecs,vocab_words,context_words)
```

Out[60]:
```
['soul',
 'kate',
 'spoke',
 'dying',
 'wonder',
 'loved',
 'creatures',
 'blame',
 'gentle',
 'old']
```

In [61]:
```
find_analogy('soil','grass','sun',vecs,vocab_words,context_words)
```

Out[61]:
```
['dark',
 'closed',
 'blue',
 'rain',
 'saw',
 'around',
 'suddenly',
 'hung',
 'stood',
 'rose']
```

In [66]:
```
find_analogy('mayor','city','president',vecs,vocab_words,context_words)
```

Out[66]:
```
['local',
 'report',
 'members',
 'state',
 'department',
 'service',
 'government',
 'board',
 'today',
 'kennedy']
```

In [ ]:
```

```