MACHINE LEARNING HW 5

K-MEANS ALGORITHM:

dataset includes $x_1 = (1,2)$, $x_2 = (2,2)$, $x_3 = (2,1)$, $x_4 = (-1,5)$
$x_5 = (-2,-1)$, $x_6 = (-1,-1)$. We want 2 clusters.

(a) $\mu_1 = x_1$, $\mu_2 = x_4$

Iteration 1:

$d(x_1,x_2)=1$ $d(x_1,x_3)=2$ $d(x_1,x_5)=6$ $d(x_1,x_6)=5$
$d(x_4,x_2)=6$ $d(x_4,x_3)=7$ $d(x_4,x_5)=7$ $d(x_4,x_6)=6$

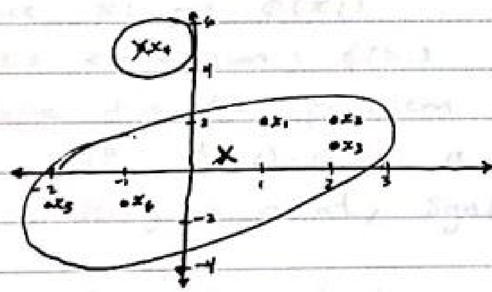
(cluster 1) $\mu_1 = (0.4, 0.6)$ and includes $\{x_1, x_2, x_3, x_5, x_6\}$
(cluster 2) $\mu_2 = (-1,5)$ and includes $\{x_4\}$

Iteration 2:

$d(\mu_1,x_1)=2$ $d(\mu_1,x_2)=3$ $d(\mu_1,x_3)=2$ $d(\mu_1,x_5)=4$ $d(\mu_1,x_6)=3$
$d(\mu_2,x_1)=5$ $d(\mu_2,x_2)=6$ $d(\mu_2,x_3)=7$ $d(\mu_2,x_5)=7$ $d(\mu_2,x_6)=6$

clusters and centers remain the same it has converged. □

(b)



clustering makes sense. x_4 is an outlier from the data an became isolated as it started as a center.

(c) To get a different clustering we could start with $\mu_1 = x_5$ and $\mu_2 = x_2$.

Iteration 1:

$d(\mu_1,x_1)=6$ $d(\mu_1,x_3)=6$ $d(\mu_1,x_4)=7$ $d(\mu_1,x_6)=1$
$d(\mu_2,x_1)=1$ $d(\mu_2,x_3)=1$ $d(\mu_2,x_4)=6$ $d(\mu_2,x_6)=6$

RESULT
$\mu_1 = (-1.5, -1)$
$\mu_2 = (1, 2.5)$

Iteration 2:

$d(\mu_1,x_1)=5.5$ $d(\mu_1,x_2)=6.5$ $d(\mu_1,x_3)=5.5$ $d(\mu_1,x_4)=6.5$ $d(\mu_1,x_5)=0.5$ $d(\mu_1,x_6)=0.$
$d(\mu_2,x_1)=0.5$ $d(\mu_2,x_2)=1.5$ $d(\mu_2,x_3)=2.5$ $d(\mu_2,x_4)=4.5$ $d(\mu_2,x_5)=6.5$ $d(\mu_2,x_6)=5$

clustering remains $\boxed{C_1 = \{x_5, x_6\} \text{ and } C_2 = \{x_1, x_2, x_3, x_4\}}$

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 math
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
from scipy.spatial import distance
```

```
[nltk_data] Downloading package brown to
[nltk_data]     C:\Users\chuck\AppData\Roaming\nltk_data...
[nltk_data]   Package brown is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\chuck\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(50):
    print (brown.words()[i],)
```

```
The
Fulton
County
Grand
Jury
said
Friday
an
investigation
of
Atlanta's
recent
primary
election
produced
``
no
evidence
''
that
any
irregularities
took
place
.
The
jury
further
said
in
term-end
presentments
that
the
City
Executive
Committee
,
which
had
over-all
charge
of
the
election
,
``
deserves
the
praise
```

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_
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)
words = []
totals = {}

## STUDENT: Your code here
# words: a python list of unique words in the document my_word_stream as the
# totals: a python dictionary, where each word is a key, and the correspondi
#         is the number of times this word appears in the document my_word_s

for word in my_word_stream:
    if word in words:
        totals[word] += 1
    else:
        totals[word] = 1
        words.append(word)

## STUDENT CODE ENDS
```

In [6]:
```
## STUDENT: Report how many times does the word "evidence" and "investigatio
print('Word "',words[10],'" appears ',totals[words[10]], ' times')
print('Word "',words[5],'" appears ',totals[words[5]], ' times')
```

```
Word " produced " appears  90  times
Word " friday " appears  60  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

for word in words:
    if totals[word] > 19:
        vocab_words.append(word)
        if totals[word] > 99:
            context_words.append(word)

## STUDENT CODE ENDS
print('Number of vocabulary words ',len(vocab_words), ';')
print('Number of context words ',len(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
    #  For instance, if window_size = 2, it means we look at w1 w2 w0 w3 w4,
    #  context woreds
    ## Output:
    #  counts: a python dictionary (of dictionaries) where counts[w0][w] ind
    #  in the context of w0 (Note: counts[w0] is also a python dictionary)

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

    for i, word in enumerate(my_word_stream):
        if word in vocab_words:
            for j in range(max(0, i-window_size), min(len(my_word_stream), i
                if i == j:
                    continue
                count_word = my_word_stream[j]
                if count_word in context_words:
                    if count_word in counts[word].keys():
                        counts[word][count_word] += 1
                    else:
                        counts[word][count_word] = 1

    ## STUDENT: Your code here


    ## End of codes
    return counts
```

In [9]:

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

```
{'obtained': 2, 'persons': 1, 'said': 3, 'much': 5, 'blue': 1, 'long': 1,
'would': 2, 'however': 4, 'several': 4, 'spent': 1, 'going': 1, 'limited':
2, 'rather': 2, 'meeting': 1, 'western': 1, 'record': 1, 'court': 2, 'littl
e': 7, 'late': 1, 'find': 1, 'good': 1, 'effort': 1, 'another': 2, 'hope':
1, 'piece': 1, 'available': 6, 'less': 1, 'nothing': 2, 'level': 1, 'adde
d': 1, 'great': 3, 'strong': 2, 'make': 3, 'necessary': 1, 'look': 2, 'exis
tence': 1, 'especially': 1, 'among': 2, 'students': 2, 'earth': 2, 'real':
1, 'turn': 1, 'enough': 2, 'industry': 1, 'stand': 1, 'property': 1, 'musi
c': 1, 'many': 2, 'bad': 1, 'question': 1, 'together': 1, 'new': 3, 'life':
1, 'power': 2, 'points': 2, 'lead': 1, 'trial': 2, 'years': 2, 'fact': 3,
'wanted': 1, 'better': 1, 'likely': 1, 'clear': 2, 'various': 1, 'written':
2, 'actually': 1, 'personal': 1, 'national': 1, 'responsibility': 1, 'suppo
rt': 2, 'ago': 1, 'provide': 1, 'deal': 1, 'us': 2, 'first': 3, 'growth':
1, 'unless': 1, 'gives': 2, 'gave': 2, 'light': 1, 'mean': 1, 'beginning':
1, 'since': 3, 'method': 1, 'taking': 1, 'number': 1, 'defense': 1, 'soon':
1, 'use': 4, 'knowledge': 1, 'general': 1, 'like': 2, 'away': 1, 'want': 1,
'come': 2, 'take': 1, 'full': 1, 'returned': 1, 'one': 2, 'recently': 2, 't
ypes': 1, 'meant': 1, 'show': 1, 'time': 1, 'material': 1, 'period': 2, 'co
nsider': 1, 'based': 1, 'evidence': 3, 'made': 1, 'difference': 2, 'straigh
t': 1, 'suggested': 1, 'religion': 1, 'either': 1, 'well': 1, 'main': 1, 'c
ourse': 2, 'taken': 1, 'given': 3, 'interest': 1, 'working': 1, 'countrie
s': 1, 'upon': 1, 'federal': 1, 'developed': 1, 'face': 1, 'growing': 1, 'e
arly': 1, 'af': 1, 'aj': 1, 'though': 4, 'unit': 1, 'important': 2, 'direc
t': 2, 'give': 2, 'side': 1, 'waiting': 1, 'present': 4, 'far': 2, 'yet':
1, 'family': 1, 'lower': 1, 'except': 1, 'numbers': 1, 'population': 1, 'ma
y': 1, 'statement': 2, 'methods': 1, 'mind': 1, 'related': 1, 'research':
1, 'cases': 1, 'hold': 1, 'larger': 1, 'problems': 2, 'found': 1, 'local':
1, 'provided': 1, 'running': 1, 'example': 1, 'amount': 1, 'order': 1, 'att
itude': 1, 'longer': 2, 'hand': 1, 'eyes': 1, 'closed': 1, 'following': 1,
'simply': 1, 'physical': 1, 'dark': 1, 'must': 2, 'world': 1, 'easy': 1, 't
hinking': 1, 'wrote': 1, 'indeed': 1, 'best': 3, 'comes': 2, 'could': 2, 'b
rown': 1, 'considered': 1, 'different': 2, 'kind': 1, 'perhaps': 1, 'recen
t': 1, 'really': 1, 'matter': 1, 'police': 1, 'anything': 1, 'sort': 1, 'ge
t': 1, 'man': 1, 'received': 1, 'outside': 1}
```

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] ind
    #  in the context of w0 (Note: counts[w0] is also a python dictionary)
    ## Output:
    #  probs: a python dictionary (of dictionaries) where probs[w0][w] indic
    #  in the context of word w0

    probs = {}

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

    for word in counts.keys():
        total = 0

        for subword in counts[word].keys():
            total += counts[word][subword]

        for subword in counts[word].keys():
            probs[word][subword] = float(counts[word][subword]) / float(tota

    ## STUDENT: Your code here

    ## End of codes
    return probs
```

In [11]:

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

```
0.010416666666666666
```

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
        diff_logs = math.log(probs[w0][w]) - math.log(context_frequency[w])
        pmi[i,j] = max(0, diff_logs)
        ## Student end of code
```

```
Computing counts and distributions
Computing pointwise mutual information
```

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

```
1.6657997172507537
```

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)](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, K):
    ## 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
    w_index = vocab_words.index(w)
    distances = []

    #calc all distances
    for i, w0 in enumerate(vocab_words):
        distances.append((distance.cityblock(vecs[w_index], vecs[i]), w0))

    #find N nearest neighbors
    distances.sort(key = lambda x: x[0])
    return distances[1:K+1]
    ## Student: code ends
```

In [18]:
```
word_NN('world',vecs,vocab_words,context_words, 5)
```

Out[18]:
```
[(7.312658394465592, 'war'),
 (7.426339216617139, 'peace'),
 (7.895204315480719, 'nations'),
 (8.085514937096962, 'history'),
 (8.094092396138727, 'nation')]
```

In [19]:
```
word_NN('learning',vecs,vocab_words,context_words, 5)
```

Out[19]:
```
[(8.62079121669431, 'necessary'),
 (8.677025570212718, 'even'),
 (8.866903951615488, 'nevertheless'),
 (8.908616457534999, 'need'),
 (8.931364662277659, 'present')]
```

In [20]:
```
word_NN('technology',vecs,vocab_words,context_words, 5)
```

Out[20]:
```
[(8.281225646607872, 'essentially'),
 (8.590205676306255, 'missiles'),
 (8.729041905155256, 'increasing'),
 (8.80162162478013, 'language'),
 (8.818157045218472, 'science')]
```

In [21]:
```
word_NN('man',vecs,vocab_words,context_words, 5)
```

Out[21]:
```
[(6.307236775303336, 'woman'),
 (7.314282450419974, 'never'),
 (7.375497015074973, 'wife'),
 (7.409964953112059, 'girl'),
 (7.482312120189896, 'son')]
```

** 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 [22]:

```
def find_analogy(A,B,C,vecs,vocab_words,context_words, K=10):
    ## 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 here
    if not((A in vocab_words) and (B in vocab_words) and (C in vocab_words))
        print("Unknown word")
        return

    ## Student: your code here
    a_index = vocab_words.index(A)
    b_index = vocab_words.index(B)
    c_index = vocab_words.index(C)

    distances = []

    #goals is the closest to (B-A) = (D-C)
    #                         D = (B-A)+C
    goal = vecs[b_index] - vecs[a_index] + vecs[c_index]

    #calc all distances
    for i, w0 in enumerate(vocab_words):
        distances.append((distance.cityblock(goal, vecs[i]), w0))

    #find N nearest neighbors
    distances.sort(key = lambda x: x[0])
    for i in range(K+1):
        print(distances[i])

    D_neighbors = word_NN(C,vecs,vocab_words,context_words, 10)
    for d in D_neighbors:
        for dist in distances[0:15]:
            if d[1] == dist[1]:
                print("Analogy Answer:", d[1])
                return d

    print("no good answers")

    return distances[0:K+1]

    ## STUDENT: your code ends
```

In [23]:
```
find_analogy('king','queen','man',vecs,vocab_words,context_words)
```

```
(8.984697846163128, 'queen')
(10.221045264540034, 'man')
(11.080216040780813, 'art')
(11.489801973996437, 'freely')
(11.529140756452565, 'memory')
(11.547423299601578, 'woman')
(11.590046334557204, 'sky')
(11.618110502114387, 'wine')
(11.622127840043412, 'shirt')
(11.627585684937296, 'red')
(11.639262294693893, 'papa')
Analogy Answer: woman
```

Out[23]:
```
(6.307236775303336, 'woman')
```

In [24]:
```
find_analogy('soil','grass','sun',vecs,vocab_words,context_words)
word_NN('sun',vecs,vocab_words,context_words, 5)
```

```
(10.976847489204651, 'sun')
(11.011307190325368, 'rain')
(11.13379857340074, 'grass')
(11.707112338738282, 'shoulders')
(11.853601337727882, 'rose')
(11.87324305720544, 'floor')
(11.88448929727884, 'closed')
(11.89065827552857, 'red')
(11.980089297445657, 'beneath')
(12.054259064471847, 'shining')
(12.054601116108604, 'suddenly')
Analogy Answer: dark
```

Out[24]:
```
[(7.7074759000459, 'light'),
 (7.739082521862044, 'stayed'),
 (7.891781704320759, 'summer'),
 (7.940921255599043, 'day'),
 (8.046904246095046, 'afternoon')]
```

In [ ]:
```

```