In [140]:
```
%%javascript
IPython.OutputArea.auto_scroll_threshold = 9999;
```

Homework 4 - Raul G. Martinez (PID: A12461871)

DSE 220: Machine Learning

Due Date: May 28, 11:59 PM

1 Instructions

A report for this Homework should be submitted on Gradescope by May 28. The code will only be evaluated, so make sure that you include your code. To secure full marks both the report and the code should be in sync and logically correct. Please only submit relevant and legible code. Even if your method does not run well, you will gain marks for comprehensiveness of your analysis as and where applicable. Please complete this homework individually.

2 Overview

The large number of English words can make language-based applications daunting. To cope with this, it is helpful to have a clustering or embedding of these words, so that words with similar meanings are clustered together, or have embedding that are close to one another.

But how can we get at the meanings of words? John Firth (1957) put it thus: You shall know a word by the company it keeps.

That is, words that tend to appear in similar contexts are likely to be related. In this assignment, you will investigate this idea by coming up with an embedding of words that is based on co-occurrence statistics.

The description here assumes you are using Python with NLTK.

1. First, download the Brown corpus (using nltk.corpus). This is a collection of text samples from a wide range of sources, with a total of over a million words. Calling brown.words() returns this text in one long list, which is useful.

In [1]:
```
import nltk
nltk.download('brown')
from nltk.corpus import brown

# get brown corpus words
brown_words = brown.words()

# print number of words
len(brown_words)
```

```
[nltk_data] Downloading package brown to /Users/gio/nltk_data...
[nltk_data]   Package brown is already up-to-date!
```

Out[1]:
```
1161192
```

2. Remove stopwords and punctuation, make everything lowercase, and count how often each word occurs. Use this to come up with two lists:

- A vocabulary V , consisting of a few thousand (e.g., 5000) of the most commonly-occurring words.
- A shorter list C of at most 1000 of the most commonly-occurring words, which we shall call context words.

In [2]:
```
import string

# make lowercase
brown_words = [i.lower() for i in brown_words]

# remove punctuations from each word
brown_words = [i.translate(str.maketrans('','',string.punctuation)) for i in brown_words]

# exclude spaces and punctuations
brown_words = [i for i in brown_words if i not in set([''] + list(string.punctuation))]

# print number of words
len(brown_words)
```

Out[2]:
```
1013319
```

In [3]:
```
from nltk.corpus import stopwords
nltk.download('stopwords')

# get stop words from nltk and remove them
stopwords = set(stopwords.words('english'))
brown_words = [i for i in brown_words if i not in stopwords]

# print number of words
len(brown_words)
```

```
[nltk_data] Downloading package stopwords to /Users/gio/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[3]:
```
539921
```

In [4]:
```
# get word frequency
word_count = {i:0 for i in set(brown_words)}
for i in brown_words:
    word_count[i]+=1

# convert to tuple and sort by count
word_count = sorted([(k,v) for k,v in word_count.items()], key=lambda x: x[1], reverse=True)

# print first ten elements of sorted tuple
word_count[:10]
```

Out[4]:
```
[('one', 3297),
 ('would', 2714),
 ('said', 1961),
 ('new', 1635),
 ('could', 1601),
 ('time', 1598),
 ('two', 1412),
 ('may', 1402),
 ('first', 1361),
 ('like', 1292)]
```

In [5]:
```
# define vocabulary V, get 5,000 most commonly-occurring words
V = [i[0] for i in word_count[:5000]]

# define list C, get 1,000 most commonly-occurring words
C = [i[0] for i in word_count[:1000]]
```

3. For each word w ∈ V , and each occurrence of it in the text stream, look at the surrounding window of four words (two before, two after):

w1 w2 w w3 w4.

Keep count of how often context words from C appear in these positions around word w. That is, for w ∈ V, c ∈ C, define

n(w, c) = # of times c occurs in a window around w.

Using these counts, construct the probability distribution Pr(c|w) of context words around w (for each w ∈ V), as well as the overall distribution Pr(c) of context words. These are distributions over C.

In [6]:

```
%%time

# find n(w,c)

import numpy as np

# define empty array with 5,000 x 1,000 elements
WordMatrix = np.zeros((5000, 1000))

# convert to sets for superior speed
set_V = set(V)
set_C = set(C)

# loop over all words from brown corpus (stream text)
for n, stream_word in enumerate(brown_words):

    # check if word is in vocabulary
    if stream_word in set_V:

        # find indices for window -2 to +2 (exclude indices from corner cases at the beginning and end)
        textStream_indices = [i for i in range(n-2, n+3) if i>=0 and i<len(brown_words) and i!=n]

        # loop over the window of four words (less than four words for some corner cases)
        for idx in textStream_indices:

            # use index to get word from stream text
            window_word = brown_words[idx]

            # update WordMatrix count if word is present on context words
            if window_word in set_C:
                WordMatrix[V.index(stream_word), C.index(window_word)]+=1

# display shape of matrix
WordMatrix.shape
```

```
CPU times: user 31.9 s, sys: 286 ms, total: 32.2 s
Wall time: 32.8 s
```

Out[6]:

```
(5000, 1000)
```

In [7]:
```
np.seterr(divide='ignore', invalid='ignore')

# find the probability distribution Pr(c|w)
Pr_c_w = WordMatrix/np.sum(WordMatrix, axis=1)[:, None]

Pr_c_w.shape
```

Out[7]: (5000, 1000)

In [8]:
```
# find the overall distribution Pr(c)
Pr_c = np.sum(WordMatrix/np.sum(WordMatrix), axis=0)

Pr_c.shape
```

Out[8]: (1000,)

4. Represent each vocabulary item w by a |C|-dimensional vector ϕ(w), whose cth coordinate is:

$$\phi(w) = \max(0, \log \Pr(c|w) / \Pr(c))$$

This is known as the (positive) pointwise mutual information, and has been quite successful in work on word embedding.

In [9]:
```
import numpy as geek

# find element-wise maximum value
Phi_w = geek.maximum(np.zeros((5000, 1000)),np.log(Pr_c_w/Pr_c))

Phi_w.shape
```

Out[9]: (5000, 1000)

5. Suppose we want a 100-dimensional representation. How would you achieve this?

Method 1: Principal Component Analysis (PCA)

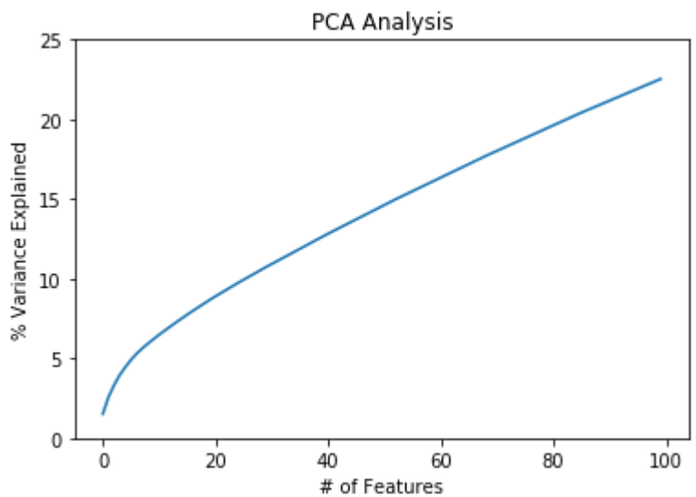
In [10]:
```
from sklearn.decomposition import PCA

# use PCA to reduce to a 100-dimensional representation
pca = PCA(n_components=100)
pca.fit(Phi_w)

# reduced data
Phi_w_transformed_PCA = pca.fit_transform(Phi_w)

Phi_w_transformed_PCA.shape
```

Out[10]:
```
(5000, 100)
```

In [11]:
```
import matplotlib.pyplot as plt

# find percentage of variance explained using PCA
perc_var = np.round(pca.explained_variance_ratio_, decimals=4)*100

# plot cummulative distribution
plt.ylabel('% Variance Explained')
plt.xlabel('# of Features')
plt.title('PCA Analysis')
plt.ylim(0,25)

print('PCA total % variance explained: {:.2f}'.format(np.sum(perc_var)))
plt.plot(np.cumsum(perc_var))
```

```
PCA total % variance explained: 22.53
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x1a29f9a710>]
```



Method 2: Singular Value Decomposition (SVD)

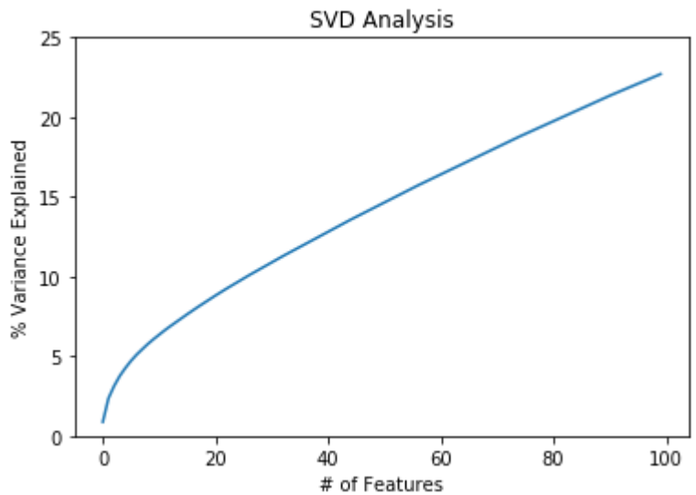
In [12]:
```
from sklearn.decomposition import TruncatedSVD

# use SVD to reduce to a 100-dimensional representation
svd = TruncatedSVD(n_components=100)
svd.fit(Phi_w)

# reduced data
Phi_w_transformed_SVD = svd.fit_transform(Phi_w)

Phi_w_transformed_SVD.shape
```

Out[12]:
```
(5000, 100)
```

In [13]:
```
# find percentage of variance explained using PCA
perc_var = np.round(svd.explained_variance_ratio_, decimals=4)*100

# plot cummulative distribution
plt.ylabel('% Variance Explained')
plt.xlabel('# of Features')
plt.title('SVD Analysis')
plt.ylim(0,25)

print('SVD total % variance explained: {:.2f}'.format(np.sum(perc_var)))
plt.plot(np.cumsum(perc_var))
```

```
SVD total % variance explained: 22.70
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x1a28d00250>]
```



6. Investigate the resulting embedding in two ways:

- Cluster the vocabulary into 100 clusters. Look them over; do they seem completely random, or is there some sense to them?
- Try finding the nearest neighbor of selected words. Do the answers make sense?

In [14]:
```
%%time

from sklearn.cluster import KMeans

# initialize k-means
kmeans_PCA = KMeans(n_clusters=100, random_state=0, init='k-means++')
kmeans_SVD = KMeans(n_clusters=100, random_state=0, init='k-means++')

# git the k-means by passing it the input
kmeans_PCA.fit(Phi_w_transformed_PCA) # method 1: PCA
kmeans_SVD.fit(Phi_w_transformed_SVD) # method 2: SVD

# retrieve labels (cluster ids) for all the inputs
labels_PCA = kmeans_PCA.labels_
labels_SVD = kmeans_SVD.labels_
```

```
CPU times: user 23.5 s, sys: 1.79 s, total: 25.3 s
Wall time: 13.3 s
```

In [15]:
```
from collections import defaultdict

# initialize dictionaries
V_clusters_PCA = defaultdict(list)
V_clusters_SVD = defaultdict(list)

# form clusters
for c in range(100):

    # add clusters inside dict as lists
    V_clusters_PCA[c] = [V[i] for i in np.where(labels_PCA == c)[0]]
    V_clusters_SVD[c] = [V[i] for i in np.where(labels_SVD == c)[0]]

# display number of clusters
len(V_clusters_PCA)
```

Out[15]:
```
100
```

In [26]:
```
# display clusters for selected words

# define vocabulary word indices
idx1, idx2 = 3563, 455

# predict using PCA
print('Embedding Method 1: PCA')
print('-----> word 1: {}'.format(V[idx1]))
Clust_word1_PCA = V_clusters_PCA[kmeans_PCA.predict([Phi_w_transformed_PCA[idx1]])[0]]
print(Clust_word1_PCA)
print('-----> word 2: {}'.format(V[idx2]))
Clust_word2_PCA = V_clusters_PCA[kmeans_PCA.predict([Phi_w_transformed_PCA[idx2]])[0]]
print(Clust_word2_PCA)

# predict using SVD
print('\nEmbedding Method 2: SVD')
print('-----> word 1: {}'.format(V[idx1]))
Clust_word1_SVD = V_clusters_SVD[kmeans_SVD.predict([Phi_w_transformed_SVD[idx1]])[0]]
print(Clust_word1_SVD)
print('-----> word 2: {}'.format(V[idx2]))
Clust_word2_SVD = V_clusters_SVD[kmeans_SVD.predict([Phi_w_transformed_SVD[idx2]])[0]]
print(Clust_word2_SVD)

# print similar words
print('\nSimilar words')
print('-----> word 1: {}'.format(V[idx1]))
print(set(Clust_word1_PCA).intersection(set(Clust_word1_SVD)))
print('-----> word 2: {}'.format(V[idx2]))
print(set(Clust_word2_PCA).intersection(set(Clust_word2_SVD)))
```

```
Method 1: PCA
-----> word 1: magnitude
['radio', 'mass', 'energy', 'source', 'produced', 'sources', 'loss', 'speed', 'plus', 'impact', 'resolution', 'associated', 'scale', 'intens
ity', 'measurements', 'foods', 'efficiency', 'comparison', 'concentration', 'essentially', 'models', 'wave', 'relative', 'atomic', 'meat',
'thickness', 'resulting', 'possibilities', 'sufficiently', 'particles', 'comparable', 'presumably', 'thyroid', 'output', '1000', 'visible',
'receiving', 'chlorine', 'missiles', 'emission', 'density', 'flux', 'proportion', 'magnitude', 'electron', 'velocity', 'estimates', 'heatin
g', 'diffusion', 'planets', 'lengths', 'availability', 'particle', 'planetary', 'input', 'biological']
-----> word 2: miles
['two', 'years', 'three', 'days', 'later', 'several', 'four', 'times', 'five', 'ago', 'six', 'minutes', 'months', 'hours', 'miles', 'hundre
d', 'ten', 'weeks', 'nearly', 'persons', 'couple', 'seven', 'eight', 'spent', 'dollars', 'thousand', 'nine', 'spend', 'twelve', 'oclock', 'e
leven', 'fourteen']

Method 2: SVD
-----> word 1: magnitude
```

```
['mass', 'produce', 'impact', 'cutting', 'resolution', 'associated', 'tension', 'intensity', 'conventional', 'foods', 'efficiency', 'extreme
ly', 'concentration', 'models', 'periods', 'wave', 'electrical', 'load', 'considerably', 'resulting', 'occur', 'possibilities', 'sufficientl
y', 'particles', 'presumably', 'introduction', 'pressures', 'thyroid', 'organic', 'contains', 'devices', 'output', 'visible', 'receiving',
'chlorine', 'emission', 'variation', 'skywave', 'density', 'fallout', 'strain', 'dimensions', 'flux', 'magnitude', 'electron', 'roots', 'vel
ocity', 'reflect', 'exposure', 'composition', 'estimates', 'heating', 'diffusion', 'subjected', 'phases', 'planets', 'crystal', 'sphere', 'f
requency', 'availability', 'loop', 'particle', 'fluid', 'protein', 'planetary', 'planet', 'input', 'fats', 'binding', 'hypothalamus', 'react
ivity', 'optical', 'nonspecific']
-----> word 2: miles
['two', 'years', 'three', 'several', 'four', 'five', 'ago', 'six', 'minutes', 'miles', 'hundred', 'ten', 'couple', 'seven', 'eight', 'dollar
s', 'thousand', 'nine', 'twenty', 'fifty', 'thirty', 'fifteen', 'twelve', 'eleven', 'forty', 'fourteen', 'twentyfive']

Similar words
-----> word 1: magnitude
{'visible', 'sufficiently', 'intensity', 'availability', 'flux', 'resulting', 'concentration', 'wave', 'resolution', 'receiving', 'velocit
y', 'planetary', 'possibilities', 'estimates', 'impact', 'particle', 'mass', 'associated', 'foods', 'heating', 'magnitude', 'density', 'elec
tron', 'diffusion', 'input', 'emission', 'presumably', 'models', 'planets', 'thyroid', 'efficiency', 'output', 'chlorine', 'particles'}
-----> word 2: miles
{'years', 'minutes', 'three', 'fourteen', 'two', 'ten', 'five', 'several', 'hundred', 'seven', 'nine', 'eleven', 'couple', 'four', 'dollar
s', 'twelve', 'thousand', 'ago', 'six', 'eight', 'miles'}
```

Answer (Q6) - Clustering the vocabulary to 100 clusters with K-Means

The clusters generated after dimensionality reduction, using PCA and SVD methods, seem to make sense and are not completely at random. To illustrate this idea two words have been chosen at random (see cell above) and their respective clusters are displayed using each method. Looking at the first word "magnitude" it seems that both methods are clustering very similarly with about 34 words overlapping between the both, and the context of the words are related mostly to science such as chemistry, biology, and physics. Likewise, the second word "miles" seems to cluster similarly with 21 words overlapping, and the context of the cluster seems to be related to time, numbers, and currency units.

In general, the 100 dimensional representation using both methods (PCA and SVD) seems to be very similar in variance metrics with a total percentage of variance explained of about 23%. When K-means is applied to the vocabulary they both create clusters with words that have similar meaning, but the clusters are not exactly the same. The variation introduced when finding the clusters doesn't really mean that one is performing better over the other, but instead that they are capturing similar information differently. As a consequence, their applications would have to be considered based on the cluster labels defined and the context behind the words being used to process the stream of text.

In [134]:
```
from sklearn.neighbors import NearestNeighbors

# try finding the nearest neighbor for 25 random words from the vocabulary

# Embedding Method 1: PCA

# find neighbors
neigh = NearestNeighbors(n_neighbors=2, metric='cosine').fit(Phi_w_transformed_PCA)
word_idx = [randrange(len(V)) for i in range(25)] # get random list of indices
distances, indices = neigh.kneighbors(Phi_w_transformed_PCA[word_idx])

# print words and NN
print('Embedding Method 1: PCA\nVocabulary, NN Prediction\n')
NN_idx = [k[1] for k in indices] # first element is the word itself, select second
for i,j in zip(word_idx, NN_idx):
    print(V[i],',',V[j])


# Embedding Method 2: SVD

# find neighbors
neigh = NearestNeighbors(n_neighbors=2, metric='cosine').fit(Phi_w_transformed_SVD)
distances, indices = neigh.kneighbors(Phi_w_transformed_SVD[word_idx])

# print words and NN
print('\nEmbedding Method 2: SVD\nVocabulary, NN Prediction\n')
NN_idx = [k[1] for k in indices] # first element is the word itself, select second
for i,j in zip(word_idx, NN_idx):
    print(V[i],',',V[j])
```

```
Embedding Method 1: PCA
Vocabulary, NN Prediction

personnel , expenditures
milligrams , beef
window , door
correct , measurement
provided , provide
mathematics , science
studio , orange
agencies , public
jew , lonely
reorganization , payments
intensity , thermal
```

folks , dare
military , increased
corps , peace
originally , publication
plenty , theres
making , made
hated , cold
romantic , lucy
damned , maggie
answer , yes
remainder , benefits
christian , religious
standard , materials
17 , 30

Embedding Method 2: SVD
Vocabulary, NN Prediction

personnel , service
milligrams , beef
window , stood
correct , must
provided , available
mathematics , science
studio , knows
agencies , public
jew , lonely
reorganization , payments
intensity , thermal
folks , nice
military , economic
corps , peace
originally , expert
plenty , theres
making , made
hated , panic
romantic , mercy
damned , maggie
answer , something
remainder , revenues
christian , religious
standard , value
17 , oct

Answer (Q6) - Finding the nearest neighbor of selected words

A total of 25 random words were selected from the vocabulary in order to apply a Nearest Neighbors (NN) algorithm using embedded representations with PCA and SVD from a Positive Pointwise Mutual Information words matrix. The NN is then found for each word and printed side-by-side with the vocabulary word (as shown in the cell above). Looking at the list of pairs is very clear that many of the words selected make sense, meaning that is likely to find them next to each other on a text stream or they have similar context. And when comparing the performance between PCA and SVD they both seem to be performing somewhat similarly; in the example above there are about 11 words out of 25 where the NN is exactly the same which equals to ~44% accuracy.

7. The Brown corpus is very small. Current work on word embedding uses data sets that are several orders of magnitude larger, but the methodology is along the same lines.

3 What to turn in

On the due date, turn in a typewritten report containing the following elements (each labeled clearly).

1. A description of your 100-dimensional embedding.

   The description should be concise and clear, and should make it obvious exactly what steps you took to obtain your word embeddings. Below, we will denote these as $\Psi(w) \in R100$, for $w \in V$. Also clarify exactly how you selected the vocabulary V and the context words C. (35 marks)

In [163]:
```
"""
Steps used for this section are described in the implementation above, and also in the report below.
"""
```

Out[163]:
```
'\nSteps used for this section are described in the implementation above, and also in the report below. \n'
```

2. Nearest neighbor results.

   Pick a collection of 25 words $w \in V$. For each w, return its nearest neighbor $w'/= w$ in V . A popular distance measure to use for this is cosine distance:

   $$1 - (\Psi(w).\Psi(w') / ||\Psi(w)|| \, ||\Psi(w')||)$$

   Here are some suggestions for words you might choose:

   communism, autumn, cigarette, pulmonary, mankind, africa, chicago, revolution, september, chemical, detergent, dictionary, storm, worship

   Do the results make any sense? You can use other distance measures apart from cosine distance to improve the results. (30 marks)

In [160]:
```
from sklearn.neighbors import NearestNeighbors
import pandas as pd

# try finding the nearest neighbor for a collection of 25 words

# define sample words and get their indices
sample_words = ['communism', 'autumn', 'cigarette', 'pulmonary', 'mankind', 'africa', 'chicago',
                'revolution', 'september', 'chemical', 'detergent', 'dictionary', 'storm', 'worship']
sample_words_idx = [V.index(i) for i in sample_words]

# fill the collection of 25 with 11 more random words
word_idx = sample_words_idx + [randrange(len(V)) for i in range(11)]

# create dataframe with vocabulary words
df = pd.DataFrame([V[i] for i in word_idx], columns=['vocabulary'])

# find NN word using different distance metrics
for d in ['cosine', 'euclidean', 'manhattan', 'chebyshev']:

    # Embedding Method 1: PCA

    # find neighbors
    neigh = NearestNeighbors(n_neighbors=2, metric=d).fit(Phi_w_transformed_PCA)
    distances, indices = neigh.kneighbors(Phi_w_transformed_PCA[word_idx])

    # add NN to dataframe
    col_name = 'PCA' + '_' + d
    df[col_name] = [V[i] for i in [k[1] for k in indices]] # second element is NN

    # Embedding Method 2: SVD

    # find neighbors
    neigh = NearestNeighbors(n_neighbors=2, metric=d).fit(Phi_w_transformed_SVD)
    distances, indices = neigh.kneighbors(Phi_w_transformed_SVD[word_idx])

    # add NN to dataframe
    col_name = 'SVD' + '_' + d
    df[col_name] = [V[i] for i in [k[1] for k in indices]] # second element is NN

# display dataframe
df
```

Out[160]:

	vocabulary	PCA_cosine	SVD_cosine	PCA_euclidean	SVD_euclidean	PCA_manhattan	SVD_manhattan	PCA_chebyshev	SVD_chebyshev

	vocabulary	PCA_cosine	SVD_cosine	PCA_euclidean	SVD_euclidean	PCA_manhattan	SVD_manhattan	PCA_chebyshev	SVD_chebyshev
0	communism	era	war	era	utopian	utopian	utopian	giant	exploration
1	autumn	storm	summer	fogg	journey	storm	time	fogg	orleans
2	cigarette	swung	suddenly	bullet	ben	haney	time	bullet	whisky
3	pulmonary	artery	artery	artery	artery	artery	artery	artery	artery
4	mankind	divine	world	divine	fatal	twentieth	fatal	inevitably	world
5	africa	asia	asia	asia	asia	asia	asia	markets	germany
6	chicago	portland	club	portland	york	portland	portland	reports	club
7	revolution	modern	modern	time	oral	suspected	oral	legislative	killing
8	september	december	december	december	december	december	december	december	june
9	chemical	drugs	drugs	drugs	drugs	instances	drugs	drugs	drugs
10	detergent	fabrics	fabrics	fabrics	fabrics	fabrics	fabrics	illustration	folklore
11	dictionary	text	text	text	text	text	text	occurrence	text
12	storm	noon	summer	noon	fogg	noon	fogg	circuit	journey
13	worship	beliefs	organized	beliefs	conception	life	beliefs	beliefs	entering
14	received	receive	washington	first	washington	first	perhaps	special	washington
15	threw	door	front	barn	foot	barn	turned	stairs	gardens
16	ridiculous	screw	ben	one	ben	one	ben	skyros	victim
17	empirical	solutions	structure	hypothalamic	hypothalamic	hypothalamic	melody	hypothalamic	continuity
18	relatives	intimate	family	coolidge	bet	coolidge	coolidge	emperor	one
19	lack	interest	need	work	need	work	personal	view	making
20	religion	political	personal	great	personal	experience	philosophy	human	rather
21	accepted	merely	beginning	one	known	known	spencer	sometimes	history
22	knees	shoulders	shoulders	shaking	shaking	shaking	shaking	wiped	suitcase
23	city	york	york	york	york	york	york	central	located
24	billion	million	million	million	million	million	million	january	11

3. Clustering.

Using the vectorial representation Ψ(.), cluster the words in V into 100 groups. Clearly specify what algorithm and distance function you using for this, and the reasons for your choices.

Look over the resulting 100 clusters. Do any of them seem even moderately coherent? Pick out a few of the best clusters and list the words in them. (35 marks)

In [162]:
```
"""
Steps used for this section are described in the implementation above, and also in the report below.
"""
```

Out[162]:
```
'\nSteps used for this section are described in the implementation above, and also in the report below. \n'
```

4. Compare the similarities and differences when using the embedding and full matrix

In [168]:
```
from sklearn.neighbors import NearestNeighbors
import pandas as pd

# try finding the nearest neighbor for a collection of 25 words

# define sample words and get their indices
sample_words = ['communism', 'autumn', 'cigarette', 'pulmonary', 'mankind', 'africa', 'chicago',
                'revolution', 'september', 'chemical', 'detergent', 'dictionary', 'storm', 'worship']
sample_words_idx = [V.index(i) for i in sample_words]

# fill the collection of 25 with 11 more random words
word_idx = sample_words_idx + [randrange(len(V)) for i in range(11)]

# create dataframe with vocabulary words
df = pd.DataFrame([V[i] for i in word_idx], columns=['vocabulary'])

# compute neighbors without embedding
for d in ['cosine', 'euclidean', 'manhattan', 'chebyshev']:

    # find neighbors
    neigh = NearestNeighbors(n_neighbors=2, metric=d).fit(Phi_w)
    distances, indices = neigh.kneighbors(Phi_w[word_idx])

    # add NN to dataframe
    col_name = 'Full-Matrix' + '_' + d
    df[col_name] = [V[i] for i in [k[1] for k in indices]]

# display dataframe
df
```

Out[168]:

	vocabulary	Full-Matrix_cosine	Full-Matrix_euclidean	Full-Matrix_manhattan	Full-Matrix_chebyshev
0	communism	utopian	one	rico	must
1	autumn	summer	one	notte	made
2	cigarette	sleeping	one	notte	back
3	pulmonary	artery	artery	artery	population
4	mankind	christ	one	rico	every
5	africa	asia	united	notte	among
6	chicago	portland	new	kansas	provide

	vocabulary	Full-Matrix_cosine	Full-Matrix_euclidean	Full-Matrix_manhattan	Full-Matrix_chebyshev
7	revolution	music	one	notte	even
8	september	july	year	rico	1
9	chemical	drugs	one	notte	given
10	detergent	fabrics	one	rico	found
11	dictionary	text	one	rico	form
12	storm	weekend	one	rico	went
13	worship	degree	one	amen	program
14	depression	mount	one	puerto	every
15	collective	involves	one	notte	trade
16	especially	many	many	notte	work
17	often	different	one	notte	would
18	involved	fact	one	rico	among
19	paula	lover	one	rico	last
20	jobs	job	one	rico	although
21	dartmouth	college	one	rico	also
22	present	important	may	notte	perhaps
23	observations	measurements	one	tetrachloride	study
24	demand	area	one	dairy	many

In this homework, you are expected to come up with three strategies (one for each task). You should to turn-in a concise report (2-3 pages) with the results of the above tasks along with a description of your final approach. You should clearly describe your approach along with all the parameters you use. The description should be enough for someone else to recreate your model/results. Along with the description, you are expected to provide the performance analysis of your model including the shortcomings (if any) and any ideas to further improve it. You should also include a brief description of all the major approaches you tried (with code on Gradescope), and the reason why you selected your current model over them.

You won't be graded on the performance of your model but you'll be graded (30%) on the comprehensiveness of your analysis. You are not expected to come up with a new algorithm (though that would be great!), but your report should justify that the approach you suggest is equivalent/better than some of the standard models discussed in the class. In essence, to secure full marks in this section, your analysis of this problem should be in-depth and thorough.

Answer for section 3 - Report

-

Description of the 100-dimensional embedding

This analysis contains over 1 million words from the NLTK Brown corpus, which is a collection of text samples from a variety of sources. To start, the corpus was read into Python as a list of raw words coming from all the combined text streams. Then, data processing and preparation was achieved by first converting all letters for each word to lower case and removing any spaces or punctuations such as commas and question marks, following by removing list elements that appear in the stopwords list from the NLTK corpus library. After removing stopwords the remaining list of words had over half a million elements, from here two more lists were generated named as Vocabulary (C) and Context words (C) containing the most commonly-occuring words, 5000 and 1000 respectively; therefore the second is a subset of the first. And the last step for data preparation was to create the co-occurance matrix with both lists described above, whose dimension is 5000 x 1000. The matrix is basically created by scanning through the listed text stream (with processed words) and finding the number of times each element in C occurs in a defined window of two elements before and after around each element in V.

Now, some natural language processing was performed by finding the positive pointwise mutual information (PPMI) on the co-ocurrance matrix. The first step was to find the probabilities for each element in C given elements in V, where the matrix dimension is the same as the original matrix with 5000 x 1000. Secondly, the probability distribution for each element in C was found, whose dimension is 1000. With this in mind, the PPMI was then be calculated by taking the element-wise maximum value between zero and the ratio of logs between the first and second probabilities calculated above. This was able to set any negative values to zero as the PPMI is only interested in understanding the mutual information between words and not how they occur individually, for example, it kept the relationship for the words "new york" despite the word "new" can appear many times individually in context with other words.

Lastly, the PPMI result found was highly dimensional with 1000 dimensions, therefore, using a lower dimensional representation was used with the intent to help model training efficiency, or in theory it could also possibly help with issues related to the curse of dimensionality. To achieve this, the analysis considered a 100-dimensional embedding where two methods are compared, namely Singular Value Decomposition (SVD) and Principal Component Analysis (PCA). After applying each of them, the transformed PPMI expressed about 23% total variance explained from the original data. The embedding was essentially expressing about a fifth of the initial variance with 10-fold decrease in features, these kind of trade-offs are decided according to the practicality of the problem.

Nearest neighbor results

Nearest neighbors (NN) is implemented above (section 3.2) by defining a list of suggested words and adding an additional 11 random words to form a collection of 25 words total, where every word is an element of the vocabulary. The model used comes from the libray sklearn.neighbors and it was processed using both methods tested for embedding as mentioned above, PCA and SVD. For each, the embedding (or transformed data) of dimension 100 was processed with the NearestNeighbors model for each of the 5000 words in the vocabulary. The parameters used are n_neighbors=2, since the first NN is the word itself the second one is taken. In addition to different distance metrics including cosine, eucledian, manhattan, and chebyshev. The results are summarized in the dataframe from section 3.2, and basically the results appear to make sense as the majority of words have similar context despite which distance metric or embedding method was used. However, is worth highlighting that there is a distinguishable pattern for this collection of words where all methods predicted the same word with the exception of "chebyshev", and this happens on rows 5, 11, 23, and 24 for the words asia, fabrics, york, and million. All in all, Is probable that some methods are better at prediction a particular context or cluster of words by finding specific hidden patterns, for this reason the parameter selection should depend on the type of application and the ability to practically apply restrictions that would make the model predict more accurately.

Clustering

The embedding for each method, PCA and SVD, was used to cluster the vocabulary (with 5000 words) into 100 clusters. The algorithm used comes from sklearn.cluster library and is called Kmeans, where the parameters used are n_clusters = 100 and init = 'k-means++'. Section 2.6 from the notebook above shows an example where two words were picked at random, "magnitude" and "miles". In general, the clusters generated for such words seem to be coherent across both embeddings tested, the first word "magnitude" clustered with other words similar in the context of science such as chemistry, biology, and physics. While the second word "miles" clustered with other words related with time, numbers, and currency units. Then, when comparing the clusters where each of the words belongs PCA clustered a total of 56 and 32 words including "magnitude" and "miles" respectively, and SVD a total of 73 and 27 in the same order. And when comparing the overlapping words pertaining to the clusters of "magnitude" and "miles", both embeddings agreed on 34 and 21 respectively. This is a very decent result considering that both embeddings have about 23% as the total percentage of variance explained. The amount of overlap between embeddings and higher relationship within the cluster might also well depend on the word chosen, in this case the word "magnitude" seems to be part of a larger cluster as it is likely to be found in a variety of places and disciplines as compared to the word "miles" that is probably only found when refering to distances or measurements.

Comparison using full matrix and reduced matrix

To keep exploring the importance of the embeddings generated, the full matrix for PPMI was used to replicate the analysis for nearest neighbors from above. The results are summarized on a dataframe in section 3.4 where it basically shows how the performance is significantly lower as compared to the embedding results in section 3.2 with PCA and SVD. For instance, the full matrix resuls when using the cosine metric for distance seem to be decent as the context between words makes sense and is comparable to the embedding results, however, the other distance metrics are completely incorrect. There is an interesting repetition of errors where words are repeated when predicting the NN, for example the words "one" and "rico" appear multiple times for the distance metrics 'eucledian' and 'manhattan'. In summary, when comparing the NN prediction between full matrix and reduced matrix the winner seems to be the reduced matrix in every scenario tested in this analysis. Is possible that methods like clustering and nearest neighbor break down at higher dimensions as the distances might become highly concentrated or the values get lost in space, these observations could definitely be tied to the curse of dimensionality. And with this in mind, the need to create an embeding before clustering is of absolute importance in order to produce a useful model with accurate predictions.

Future steps

Lastly, to further explore this analysis there are a few things to consider around data preparation, domain knowledge, and statistics to bring better insight on the results. First, some restrictions in the vocabulary might be useful in order to elimiate added noise to the predictions. For example some words from the vocabulary appear as numerical values or text numbers (i.e. one), and in the context of words these can possibly appear in many places without any specific mutual information of interest between words. Another example would be to reduce the number of words by considering linguistics, words that have similar meaning, or known noise words that might induce negative interactions. Secondly, after data preparation is considered and clustering is performed, it would be useful to talk to a domain expert about each of the clusters learned in order to identify the nature of each group and whether they make sense to belong together or not. In addition, to keep exploring the clusters it would be fundamental to count and quantify que occurence of each word as multiple embedding methods (i.e. PCA) and parameters (i.e. cosine distance) are tested and implemented. And lastly, more work can be done in understanding the words that are clustered near the boundaries among other clusters as they might provide useful insights when trying to understand relationships, rank importance, or being able to discriminate with higher confidence.

In [ ]: