Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says "Write your answer here.", as well as your name and NetID below:

In [1]:

```
NAME = "Yuhang Chen"
NetID = "yuhangc3"
```

Submission Instructions

- This lab has two components: the coding component as well as a report (text written) component.
- The coding component comprises Q1.1, Q1.2, Q1.3. You will be writing the code for these questions in the file lab4.py. The grading for this component will take place through the gradescope autograder by submitting the lab4.py file to the **NLP Lab4 Code** assignment on Gradescope.
- The report component comprises Q1.4, Q1.5, Q2.1, Q2.2, Q2.3, Q2.4, Q2.5, Q2.6, Q2.7, Q2.8.
- The report component comprises three kinds of questions: Code+Plot, Code+Written and Written. For the Code+Plot (e.g Q1.4) type you will be graded for the code you write and the corresponding plot produce. For the Code+Written(e.g Q2.7) type you will be graded for the code you write and the corresponding written response. For the Written(e.g Q2.8) type you will be graded for the written response. You can enter the written responses in the corresponding space provided in the jupyter notebook itself, by editing the corresponding markdown cell. Make sure you fill in any place that says "Write your answer here."
- To submit the questions that comprise the report section, you can either convert the jupyter notebook to a pdf by following one of two methods described in the next cell (choose whichever method that produces the best pdf render) or you can take screenshots of the corresponding section of the jupyter notebook, paste the screenshots onto a document (eg .doc) file and then convert to pdf. For the Code+Written and Code+Plot questions, make sure the code cell as well as the written/plot cell is included in the screenshot. For the Written questions, you can only include the written response in your screenshot.
- Once you have prepared the report using either the direct PDF or the screenshot route, submit the corresponding file to the Gradescope assignment called **NLP Lab4 Report**.
- **It is extremely important that you tag the corresponding pages of the report to the appropriate question outline on gradescope. Failure to tag the appropriate/correct pages can lead to deduction of points.**

Converting Jupyter Notebook to PDF [Method 1]

1. Click the Save button at the top of the Jupyter Notebook.
2. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of all cells).
3. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.

4. Once you've rerun everything, select File -> Download as -> PDF via LaTeX
5. Look at the PDF file and make sure all your solutions are there, displayed correctly.

Converting Jupyter Notebook to PDF [Method 2]

1. Click the Save button at the top of the Jupyter Notebook.
2. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of all cells).
3. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.
4. Once you've rerun everything, run Cmd+P (Ctrl+P on windows) -> Save as PDF.
5. Look at the PDF file and make sure all your solutions are there, displayed correctly.

Lab4: Word Vectors

In [2]:

```
# All Import Statements Defined Here
# Note: Do not add to this list.
# All the dependencies you need, can be installed by running .
# Make sure to have scikit-learn version 0.21.3, otherwise you might run into
# I found it best to create a new conda environment with scikit-learn 0.21.3 i
# conda create -n lab4_env python=3 scikit-learn==0.21.3
# ----------------

import sys
assert sys.version_info[0]==3
assert sys.version_info[1] >= 5

from gensim.models import KeyedVectors
from gensim.test.utils import datapath
import pprint
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10, 5]
import nltk
nltk.download('reuters')
from nltk.corpus import reuters
import numpy as np
import random
import scipy as sp
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import PCA

START_TOKEN = '<START>'
END_TOKEN = '<END>'

np.random.seed(0)
random.seed(0)
# ----------------
```

```
C:\ProgramData\Anaconda3\lib\site-packages\gensim\similarities
\__init__.py:15: UserWarning: The gensim.similarities.levenshte
in submodule is disabled, because the optional Levenshtein pack
age <https://pypi.org/project/python-Levenshtein/> is unavailab
le. Install Levenhstein (e.g. `pip install python-Levenshtein`)
to suppress this warning.
  warnings.warn(msg)
```

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

In [3]:

```
from importlib import reload
import lab4
```

Word Vectors

Word Vectors are often used as a fundamental component for downstream NLP tasks, e.g. question answering, text generation, translation, etc., so it is important to build some intuitions as to their strengths and weaknesses. Here, you will explore two types of word vectors: those derived from *co-occurrence matrices*, and those derived via *word2vec*.

Assignment Notes: Please make sure to save the notebook as you go along. Submission Instructions are located at the bottom of the notebook.

Note on Terminology: The terms "word vectors" and "word embeddings" are often used interchangeably. The term "embedding" refers to the fact that we are encoding aspects of a word's meaning in a lower dimensional space. As Wikipedia (https://en.wikipedia.org/wiki/Word_embedding) states, "*conceptually it involves a mathematical embedding from a space with one dimension per word to a continuous vector space with a much lower dimension*".

Part 1: Count-Based Word Vectors (50 points)

Most word vector models start from the following idea:

You shall know a word by the company it keeps (Firth, J. R. 1957:11 (https://en.wikipedia.org/wiki/John_Rupert_Firth))

Many word vector implementations are driven by the idea that similar words, i.e., (near) synonyms, will be used in similar contexts. As a result, similar words will often be spoken or written along with a shared subset of words, i.e., contexts. By examining these contexts, we can try to develop embeddings for our words. With this intuition in mind, many "old school" approaches to constructing word vectors relied on word counts. Here we elaborate upon one of those strategies, *co-occurrence matrices* (for more information, see here (https://medium.com/data-science-group-iitr/word-embedding-2d05d270b285)).

Co-Occurrence

A co-occurrence matrix counts how often things co-occur in some environment. Given some word w_i occurring in the document, we consider the *context window* surrounding w_i. Supposing our fixed window size is n, then this is the n preceding and n subsequent words in that document, i.e. words $w_{i-n} \dots w_{i-1}$ and $w_{i+1} \dots w_{i+n}$ We build a *co-occurrence matrix* M, which is a symmetric word-by-word matrix in which M_{ij} is the number of times w_j appears inside w_i's window.

Example: Co-Occurrence with Fixed Window of n=1:

Document 1: "all that glitters is not gold"

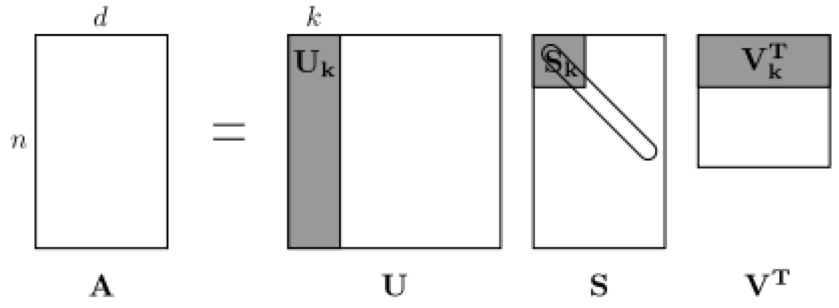
Document 2: "all is well that ends well"

*	START	all	that	glitters	is	not	gold	well	ends	END
START	0	2	0	0	0	0	0	0	0	0
all	2	0	1	0	1	0	0	0	0	0
that	0	1	0	1	0	0	0	1	1	0
glitters	0	0	1	0	1	0	0	0	0	0
is	0	1	0	1	0	1	0	1	0	0
not	0	0	0	0	1	0	1	0	0	0
gold	0	0	0	0	0	1	0	0	0	1

*	START	all	that	glitters	is	not	gold	well	ends	END
well	0	0	1	0	1	0	0	0	1	1
ends	0	0	1	0	0	0	0	1	0	0
END	0	0	0	0	0	0	1	1	0	0

Note: In NLP, we often add START and END tokens to represent the beginning and end of sentences, paragraphs or documents. In thise case we imagine START and END tokens encapsulating each document, e.g., "START All that glitters is not gold END", and include these tokens in our co-occurrence counts.

The rows (or columns) of this matrix provide one type of word vectors (those based on word-word co-occurrence), but the vectors will be large in general (linear in the number of distinct words in a corpus). Thus, our next step is to run *dimensionality reduction*. In particular, we will run *SVD (Singular Value Decomposition)*, which is a kind of generalized *PCA (Principal Components Analysis)* to select the top k principal components. Here's a visualization of dimensionality reduction with SVD. In this picture our co-occurrence matrix is A with n rows corresponding to n words. We obtain a full matrix decomposition, with the singular values ordered in the diagonal S matrix, and our new, shorter length-k word vectors in U_k.



This reduced-dimensionality co-occurrence representation preserves semantic relationships between words, e.g. *doctor* and *hospital* will be closer than *doctor* and *dog*.

Notes: If you can barely remember what an eigenvalue is, here's a slow, friendly introduction to SVD (https://davetang.org/file/Singular_Value_Decomposition_Tutorial.pdf). Though, for the purpose of this class, you only need to know how to extract the k-dimensional embeddings by utilizing pre-programmed implementations of these algorithms from the numpy, scipy, or sklearn python packages. In practice, it is challenging to apply full SVD to large corpora because of the memory needed to perform PCA or SVD. However, if you only want the top k vector components for relatively small k — known as *Truncated SVD (https://en.wikipedia.org/wiki/Singular_value_decomposition#Truncated_SVD)* — then there are reasonably scalable techniques to compute those iteratively.

Plotting Co-Occurrence Word Embeddings

Here, we will be using the Reuters (business and financial news) corpus. If you haven't run the import cell at the top of this page, please run it now (click it and press SHIFT-RETURN). The corpus consists of 10,788 news documents totaling 1.3 million words. These documents span 90 categories and are split into train and test. For more details, please see https://www.nltk.org/book/ch02.html (https://www.nltk.org/book/ch02.html). We provide a `read_corpus` function below that pulls out only articles from the "crude" (i.e. news articles about oil, gas, etc.) category. The function also adds START and END tokens to each of the documents, and lowercases words. You do **not** have to perform any other kind of pre-processing.

In [4]:

```
def read_corpus(category="crude"):
    """ Read files from the specified Reuter's category.
        Params:
            category (string): category name
        Return:
            list of lists, with words from each of the processed files
    """
    files = reuters.fileids(category)
    return [[START_TOKEN] + [w.lower() for w in list(reuters.words(f))] + [E
```

Let's have a look what these documents are like….

In [5]:

```
reuters_corpus = read_corpus()
pprint.pprint(reuters_corpus[:3], compact=True, width=100)
```

```
[['<START>', 'japan', 'to', 'revise', 'long', '-', 'term', 'energy', 'demand', 'downwards', 'the',
  'ministry', 'of', 'international', 'trade', 'and', 'industry', '(', 'miti', ')', 'will', 'revise',
  'its', 'long', '-', 'term', 'energy', 'supply', '/', 'demand', 'outlook', 'by', 'august', 'to',
  'meet', 'a', 'forecast', 'downtrend', 'in', 'japanese', 'energy', 'demand', ',', 'ministry',
  'officials', 'said', '.', 'miti', 'is', 'expected', 'to', 'lower', 'the', 'projection', 'for',
  'primary', 'energy', 'supplies', 'in', 'the', 'year', '2000', 'to', '550', 'mln', 'kilolitres',
  '(', 'kl', ')', 'from', '600', 'mln', ',', 'they', 'said', '.', 'the', 'decision', 'follows',
  'the', 'emergence', 'of', 'structural', 'changes', 'in', 'japanese', 'industry', 'following',
  'the', 'rise', 'in', 'the', 'value', 'of', 'the', 'yen', 'and', 'a', 'decline', 'in', 'domestic',
  'electric', 'power', 'demand', '.', 'miti', 'is', 'planning', 'to', 'work', 'out', 'a', 'revised',
  'energy', 'supply', '/', 'demand', 'outlook', 'through', 'deliberations', 'of', 'committee',
  'meetings', 'of', 'the', 'agency', 'of', 'natural', 'resources', 'and', 'energy', ',', 'the',
  'officials', 'said', '.', 'they', 'said', 'miti', 'will', 'also', 'review', 'the', 'breakdown',
  'of', 'energy', 'supply', 'sources', ',', 'including', 'oil', ',', 'nuclear', ',', 'coal', 'and',
  'natural', 'gas', '.', 'nuclear', 'energy', 'provided', 'the', 'bulk', 'of', 'japan', "'", 's',
  'electric', 'power', 'in', 'the', 'fiscal', 'year', 'ended', 'march', '31', ',', 'supplying',
  'an', 'estimated', '27', 'pct', 'on', 'a', 'kilowatt', '/', 'hour', 'basis', ',', 'followed',
  'by', 'oil', '(', '23', 'pct', ')', 'and', 'liquefied', 'natural', 'gas', '(', '21', 'pct', '),',
  'they', 'noted', '.', '<END>'],
 ['<START>', 'energy', '/', 'u', '.', 's', '.', 'petrochemical', 'industry', 'cheap', 'oil',
  'feedstocks', ',', 'the', 'weakened', 'u', '.', 's', '.', 'dollar', 'and', 'a', 'plant',
  'utilization', 'rate', 'approaching', '90', 'pct', 'will', 'propel', 'the', 'streamlined', 'u',
  '.', 's', '.', 'petrochemical', 'industry', 'to', 'record', 'profits', 'this', 'year', ',',
  'with', 'growth', 'expected', 'through', 'at', 'least', '1990', ',', 'major', 'company',
  'executives', 'predicted', '.', 'this', 'bullish', 'outlook', 'for', 'chemical', 'manufacturing',
  'and', 'an', 'industrywide', 'move', 'to', 'shed', 'unrelated', 'businesses', 'has', 'prompted',
  'gaf', 'corp', '&', 'lt', ';', 'gaf', '>,', 'privately', '-', 'held', 'cain', 'chemical', 'inc',
  ',', 'and', 'other', 'firms', 'to', 'aggressively', 'seek', 'acquisitions', 'of', 'petrochemical',
  'plants', '.', 'oil', 'companies', 'such', 'as', 'ashlan
d', 'oil', 'inc', '&', 'lt', ';', 'ash',
  '>,', 'the', 'kentucky', '-', 'based', 'oil', 'refiner',
'and', 'marketer', ',', 'are', 'also',
  'shopping', 'for', 'money', '-', 'making', 'petrochemica
l', 'businesses', 'to', 'buy', '.', '"',
  'i', 'see', 'us', 'poised', 'at', 'the', 'threshold', 'o
f', 'a', 'golden', 'period', ',"', 'said',
  'paul', 'oreffice', ',', 'chairman', 'of', 'giant', 'dow',
'chemical', 'co', '&', 'lt', ';',
  'dow', '>,', 'adding', ',', ';"', 'there', "'", 's', 'no',
'major', 'plant', 'capacity', 'being',
  'added', 'around', 'the', 'world', 'now', '.', 'the', 'who
le', 'game', 'is', 'bringing', 'out',
  'new', 'products', 'and', 'improving', 'the', 'old', 'one
s', '."', 'analysts', 'say', 'the',
  'chemical', 'industry', "'", 's', 'biggest', 'customers',
',', 'automobile', 'manufacturers',
  'and', 'home', 'builders', 'that', 'use', 'a', 'lot', 'o
f', 'paints', 'and', 'plastics', ',',
  'are', 'expected', 'to', 'buy', 'quantities', 'this', 'yea
r', '.', 'u', '.', 's', '.',
  'petrochemical', 'plants', 'are', 'currently', 'operatin
g', 'at', 'about', '90', 'pct',
  'capacity', ',', 'reflecting', 'tighter', 'supply', 'tha
t', 'could', 'hike', 'product', 'prices',
  'by', '30', 'to', '40', 'pct', 'this', 'year', ',', 'sai
d', 'john', 'dosher', ',', 'managing',
  'director', 'of', 'pace', 'consultants', 'inc', 'of', 'hou
ston', '.', 'demand', 'for', 'some',
  'products', 'such', 'as', 'styrene', 'could', 'push', 'pro
fit', 'margins', 'up', 'by', 'as',
  'much', 'as', '300', 'pct', ',', 'he', 'said', '.', 'oreff
ice', ',', 'speaking', 'at', 'a',
  'meeting', 'of', 'chemical', 'engineers', 'in', 'houston',
',', 'said', 'dow', 'would', 'easily',
  'top', 'the', '741', 'mln', 'dlrs', 'it', 'earned', 'las
t', 'year', 'and', 'predicted', 'it',
  'would', 'have', 'the', 'best', 'year', 'in', 'its', 'hist
ory', '.', 'in', '1985', ',', 'when',
  'oil', 'prices', 'were', 'still', 'above', '25', 'dlrs',
'a', 'barrel', 'and', 'chemical',
  'exports', 'were', 'adversely', 'affected', 'by', 'the',
'strong', 'u', '.', 's', '.', 'dollar',
  ',', 'dow', 'had', 'profits', 'of', '58', 'mln', 'dlrs',
'.', ';"', 'i', 'believe', 'the',
  'entire', 'chemical', 'industry', 'is', 'headed', 'for',
'a', 'record', 'year', 'or', 'close',
  'to', 'it', ',"', 'oreffice', 'said', '.', 'gaf', 'chairma
n', 'samuel', 'heyman', 'estimated',
  'that', 'the', 'u', '.', 's', '.', 'chemical', 'industry',
'would', 'report', 'a', '20', 'pct',
  'gain', 'in', 'profits', 'during', '1987', '.', 'last', 'y
ear', ',', 'the', 'domestic',
  'industry', 'earned', 'a', 'total', 'of', '13', 'billion',
'dlrs', ',', 'a', '54', 'pct', 'leap',
  'from', '1985', '.', 'the', 'turn', 'in', 'the', 'fortune
s', 'of', 'the', 'once', '-', 'sickly',
  'chemical', 'industry', 'has', 'been', 'brought', 'about',
'by', 'a', 'combination', 'of', 'luck',
  'and', 'planning', ',', 'said', 'pace', "'", 's', 'john',
'dosher', '.', 'dosher', 'said', 'last',
  'year', "'", 's', 'fall', 'in', 'oil', 'prices', 'made',
'feedstocks', 'dramatically', 'cheaper',
  'and', 'at', 'the', 'same', 'time', 'the', 'american', 'do
llar', 'was', 'weakening', 'against',
  'foreign', 'currencies', '.', 'that', 'helped', 'boost',
'u', '.', 's', '.', 'chemical',
  'exports', '.', 'also', 'helping', 'to', 'bring', 'suppl
y', 'and', 'demand', 'into', 'balance',
  'has', 'been', 'the', 'gradual', 'market', 'absorption',
'of', 'the', 'extra', 'chemical',
  'manufacturing', 'capacity', 'created', 'by', 'middle', 'e
astern', 'oil', 'producers', 'in',
  'the', 'early', '1980s', '.', 'finally', ',', 'virtually',
'all', 'major', 'u', '.', 's', '.',
  'chemical', 'manufacturers', 'have', 'embarked', 'on', 'a
n', 'extensive', 'corporate',
  'restructuring', 'program', 'to', 'mothball', 'inefficien
t', 'plants', ',', 'trim', 'the',
  'payroll', 'and', 'eliminate', 'unrelated', 'businesses',
'.', 'the', 'restructuring', 'touched',
  'off', 'a', 'flurry', 'of', 'friendly', 'and', 'hostile',
'takeover', 'attempts', '.', 'gaf', ',',
  'which', 'made', 'an', 'unsuccessful', 'attempt', 'in', '1
985', 'to', 'acquire', 'union',
  'carbide', 'corp', '&', 'lt', ';', 'uk', '>,', 'recently',
'offered', 'three', 'billion', 'dlrs',
  'for', 'borg', 'warner', 'corp', '&', 'lt', ';', 'bor',
'>,', 'a', 'chicago', 'manufacturer',
  'of', 'plastics', 'and', 'chemicals', '.', 'another', 'ind
ustry', 'powerhouse', ',', 'w', '.',
  'r', '.', 'grace', '&', 'lt', ';', 'gra', '>', 'has', 'div
ested', 'its', 'retailing', ',',
  'restaurant', 'and', 'fertilizer', 'businesses', 'to', 'ra
ise', 'cash', 'for', 'chemical',
  'acquisitions', '.', 'but', 'some', 'experts', 'worry', 't
hat', 'the', 'chemical', 'industry',
  'may', 'be', 'headed', 'for', 'trouble', 'if', 'companie
s', 'continue', 'turning', 'their',
  'back', 'on', 'the', 'manufacturing', 'of', 'staple', 'pet
rochemical', 'commodities', ',', 'such',
  'as', 'ethylene', ',', 'in', 'favor', 'of', 'more', 'profi
table', 'specialty', 'chemicals',
  'that', 'are', 'custom', '-', 'designed', 'for', 'a', 'sma
ll', 'group', 'of', 'buyers', '.', '"',
  'companies', 'like', 'dupont', '&', 'lt', ';', 'dd', '>',
'and', 'monsanto', 'co', '&', 'lt', ';',
  'mtc', '>', 'spent', 'the', 'past', 'two', 'or', 'three',
'years', 'trying', 'to', 'get', 'out',
  'of', 'the', 'commodity', 'chemical', 'business', 'in', 'r
eaction', 'to', 'how', 'badly', 'the',
  'market', 'had', 'deteriorated', ',"', 'dosher', 'said',
'.', '"', 'but', 'i', 'think', 'they',
  'will', 'eventually', 'kill', 'the', 'margins', 'on', 'th
e', 'profitable', 'chemicals', 'in',
  'the', 'niche', 'market', '."', 'some', 'top', 'chemical',
'executives', 'share', 'the',
  'concern', '.', '"', 'the', 'challenge', 'for', 'our', 'in
dustry', 'is', 'to', 'keep', 'from',
  'getting', 'carried', 'away', 'and', 'repeating', 'past',
'mistakes', ',"', 'gaf', "'", 's',
  'heyman', 'cautioned', '.', '"', 'the', 'shift', 'from',
 'commodity', 'chemicals', 'may', 'be',
  'ill', '-', 'advised', '.', 'specialty', 'businesses', 'd
o', 'not', 'stay', 'special', 'long',
  '."', 'houston', '-', 'based', 'cain', 'chemical', ',', 'c
reated', 'this', 'month', 'by', 'the',
  'sterling', 'investment', 'banking', 'group', ',', 'believ
es', 'it', 'can', 'generate', '700',
  'mln', 'dlrs', 'in', 'annual', 'sales', 'by', 'bucking',
 'the', 'industry', 'trend', '.',
  'chairman', 'gordon', 'cain', ',', 'who', 'previously', 'l
ed', 'a', 'leveraged', 'buyout', 'of',
  'dupont', "'", 's', 'conoco', 'inc', "'", 's', 'chemical',
 'business', ',', 'has', 'spent', '1',
  '.', '1', 'billion', 'dlrs', 'since', 'january', 'to', 'bu
y', 'seven', 'petrochemical', 'plants',
  'along', 'the', 'texas', 'gulf', 'coast', '.', 'the', 'pla
nts', 'produce', 'only', 'basic',
  'commodity', 'petrochemicals', 'that', 'are', 'the', 'buil
ding', 'blocks', 'of', 'specialty',
  'products', '.', '"', 'this', 'kind', 'of', 'commodity',
 'chemical', 'business', 'will', 'never',
  'be', 'a', 'glamorous', ',', 'high', '-', 'margin', 'busin
ess', ',"', 'cain', 'said', ',',
  'adding', 'that', 'demand', 'is', 'expected', 'to', 'gro
w', 'by', 'about', 'three', 'pct',
  'annually', '.', 'garo', 'armen', ',', 'an', 'analyst', 'w
ith', 'dean', 'witter', 'reynolds', ',',
  'said', 'chemical', 'makers', 'have', 'also', 'benefitte
d', 'by', 'increasing', 'demand', 'for',
  'plastics', 'as', 'prices', 'become', 'more', 'competitiv
e', 'with', 'aluminum', ',', 'wood',
  'and', 'steel', 'products', '.', 'armen', 'estimated', 'th
e', 'upturn', 'in', 'the', 'chemical',
  'business', 'could', 'last', 'as', 'long', 'as', 'four',
 'or', 'five', 'years', ',', 'provided',
  'the', 'u', '.', 's', '.', 'economy', 'continues', 'its',
 'modest', 'rate', 'of', 'growth', '.',
  '<END>'],
 ['<START>', 'turkey', 'calls', 'for', 'dialogue', 'to', 'so
lve', 'dispute', 'turkey', 'said',
  'today', 'its', 'disputes', 'with', 'greece', ',', 'includ
ing', 'rights', 'on', 'the',
  'continental', 'shelf', 'in', 'the', 'aegean', 'sea', ',',
 'should', 'be', 'solved', 'through',
  'negotiations', '.', 'a', 'foreign', 'ministry', 'statemen
t', 'said', 'the', 'latest', 'crisis',
  'between', 'the', 'two', 'nato', 'members', 'stemmed', 'fr
om', 'the', 'continental', 'shelf',
  'dispute', 'and', 'an', 'agreement', 'on', 'this', 'issu
e', 'would', 'effect', 'the', 'security',
  ',', 'economy', 'and', 'other', 'rights', 'of', 'both', 'c
ountries', '.', '"', 'as', 'the',
  'issue', 'is', 'basicly', 'political', ',', 'a', 'solutio
n', 'can', 'only', 'be', 'found', 'by',
  'bilateral', 'negotiations', ',"', 'the', 'statement', 'sa
id', '.', 'greece', 'has', 'repeatedly',
  'said', 'the', 'issue', 'was', 'legal', 'and', 'could', 'b
e', 'solved', 'at', 'the',
  'international', 'court', 'of', 'justice', '.', 'the', 'tw
o', 'countries', 'approached', 'armed',
  'confrontation', 'last', 'month', 'after', 'greece', 'anno
unced', 'it', 'planned', 'oil',
  'exploration', 'work', 'in', 'the', 'aegean', 'and', 'turk
ey', 'said', 'it', 'would', 'also',
  'search', 'for', 'oil', '.', 'a', 'face', '-', 'off', 'wa
s', 'averted', 'when', 'turkey',
  'confined', 'its', 'research', 'to', 'territorrial', 'wate
rs', '.', '"', 'the', 'latest',
  'crises', 'created', 'an', 'historic', 'opportunity', 't
o', 'solve', 'the', 'disputes', 'between',
  'the', 'two', 'countries', ',"', 'the', 'foreign', 'minist
ry', 'statement', 'said', '.', 'turkey',
  "'", 's', 'ambassador', 'in', 'athens', ',', 'nazmi', 'aki
man', ',', 'was', 'due', 'to', 'meet',
  'prime', 'minister', 'andreas', 'papandreou', 'today', 'fo
r', 'the', 'greek', 'reply', 'to', 'a',
  'message', 'sent', 'last', 'week', 'by', 'turkish', 'prim
e', 'minister', 'turgut', 'ozal', '.',
  'the', 'contents', 'of', 'the', 'message', 'were', 'not',
'disclosed', '.', '<END>']]
```

Question 1.1: Implement `distinct_words` [code] (5 points)

Write a method to work out the distinct words (word types) that occur in the corpus. You can do this with `for` loops, but it's more efficient to do it with Python list comprehensions. In particular, this (https://coderwall.com/p/rcmaea/flatten-a-list-of-lists-in-one-line-in-python) may be useful to flatten a list of lists. If you're not familiar with Python list comprehensions in general, here's more information (https://python-3-patterns-idioms-test.readthedocs.io/en/latest/Comprehensions.html).

You may find it useful to use Python sets (https://www.w3schools.com/python/python_sets.asp) to remove duplicate words.

In [6]:

```
reload(lab4);
```

In [7]:

```
# ---------------------
# Run this sanity check
# Note that this not an exhaustive check for correctness.
# ---------------------

# Define toy corpus
test_corpus = ["START All that glitters isn't gold END".split(" "), "START All
test_corpus_words, num_corpus_words = lab4.distinct_words(test_corpus)

# Correct answers
ans_test_corpus_words = sorted(list(set(["START", "All", "ends", "that", "gold
ans_num_corpus_words = len(ans_test_corpus_words)

# Test correct number of words
assert(num_corpus_words == ans_num_corpus_words), "Incorrect number of distin

# Test correct words
assert (test_corpus_words == ans_test_corpus_words), "Incorrect corpus_words.

# Print Success
print ("-" * 80)
print("Passed All Tests!")
print ("-" * 80)
```

```
----------------------------------------------------------------
----------------
Passed All Tests!
----------------------------------------------------------------
----------------
```

Question 1.2: Implement `compute_co_occurrence_matrix` [code] (15 points)

Write a method that constructs a co-occurrence matrix for a certain window-size n (with a default of 4), considering words n before and n after the word in the center of the window. Here, we start to use `numpy (np)` to represent vectors, matrices, and tensors.

In [8]:

```
reload(lab4);
```

In [9]:

```
# ---------------------
# Run this sanity check
# Note that this is not an exhaustive check for correctness.
# ---------------------

# Define toy corpus and get student's co-occurrence matrix
test_corpus = ["START All that glitters isn't gold END".split(" "), "START All
M_test, word2Ind_test = lab4.compute_co_occurrence_matrix(test_corpus, window_

# Correct M and word2Ind
M_test_ans = np.array(
    [[0., 0., 0., 1., 0., 0., 0., 0., 1., 0.,],
     [0., 0., 0., 1., 0., 0., 0., 0., 0., 1.,],
     [0., 0., 0., 0., 0., 0., 1., 0., 0., 1.,],
     [1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,],
     [0., 0., 0., 0., 0., 0., 0., 0., 1., 1.,],
     [0., 0., 0., 0., 0., 0., 0., 1., 1., 0.,],
     [0., 0., 1., 0., 0., 0., 0., 1., 0., 0.,],
     [0., 0., 0., 0., 0., 1., 1., 0., 0., 0.,],
     [1., 0., 0., 0., 1., 1., 0., 0., 0., 1.,],
     [0., 1., 1., 0., 1., 0., 0., 0., 1., 0.,]]
)
word2Ind_ans = {'All': 0, "All's": 1, 'END': 2, 'START': 3, 'ends': 4, 'glitte

# Test correct word2Ind
assert (word2Ind_ans == word2Ind_test), "Your word2Ind is incorrect:\nCorrect

# Test correct M shape
assert (M_test.shape == M_test_ans.shape), "M matrix has incorrect shape.\nCo

# Test correct M values
for w1 in word2Ind_ans.keys():
    idx1 = word2Ind_ans[w1]
    for w2 in word2Ind_ans.keys():
        idx2 = word2Ind_ans[w2]
        student = M_test[idx1, idx2]
        correct = M_test_ans[idx1, idx2]
        if student != correct:
            print("Correct M:")
            print(M_test_ans)
            print("Your M: ")
            print(M_test)
            raise AssertionError("Incorrect count at index ({}, {})=({}, {})

# Print Success
print ("-" * 80)
print("Passed All Tests!")
print ("-" * 80)
```

```
--------------------------------------------------------------
------------------
Passed All Tests!
--------------------------------------------------------------
------------------
```

Question 1.3: Implement `reduce_to_k_dim` **[code] (10 point)**

Construct a method that performs dimensionality reduction on the matrix to produce k-dimensional embeddings. Use SVD to take the top k components and produce a new matrix of k-dimensional embeddings.

Note: All of numpy, scipy, and scikit-learn (`sklearn`) provide *some* implementation of SVD, but only scipy and sklearn provide an implementation of Truncated SVD, and only sklearn provides an efficient randomized algorithm for calculating large-scale Truncated SVD. So please use sklearn.decomposition.TruncatedSVD (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html).

In [10]:

```
reload(lab4);
```

In [11]:

```
# ---------------------
# Run this sanity check
# Note that this not an exhaustive check for correctness
# In fact we only check that your M_reduced has the right dimensions.
# ---------------------

# Define toy corpus and run student code
test_corpus = ["START All that glitters isn't gold END".split(" "), "START All
M_test, word2Ind_test = lab4.compute_co_occurrence_matrix(test_corpus, window_
M_test_reduced = lab4.reduce_to_k_dim(M_test, k=2)

# Test proper dimensions
assert (M_test_reduced.shape[0] == 10), "M_reduced has {} rows; should have {
assert (M_test_reduced.shape[1] == 2), "M_reduced has {} columns; should have

# Print Success
print ("-" * 80)
print("Passed All Tests!")
print ("-" * 80)
```

```
Running Truncated SVD over 10 words...
Done.
----------------------------------------------------------------
----------------
Passed All Tests!
----------------------------------------------------------------
----------------
```

Question 1.4: Implement `plot_embeddings` [code+plot] (10 point)

Here you will write a function to plot a set of 2D vectors in 2D space. For graphs, we will use Matplotlib (`plt`).

For this example, you may find it useful to adapt this code (https://www.pythonmembers.club/2018/05/08/matplotlib-scatter-plot-annotate-set-text-at-label-each-point/). In the future, a good way to make a plot is to look at the Matplotlib gallery (https://matplotlib.org/gallery/index.html), find a plot that looks somewhat like what you want, and adapt the code they give.

While submitting this question for the report, include the code cell as well as the corresponding plot produced by your code.

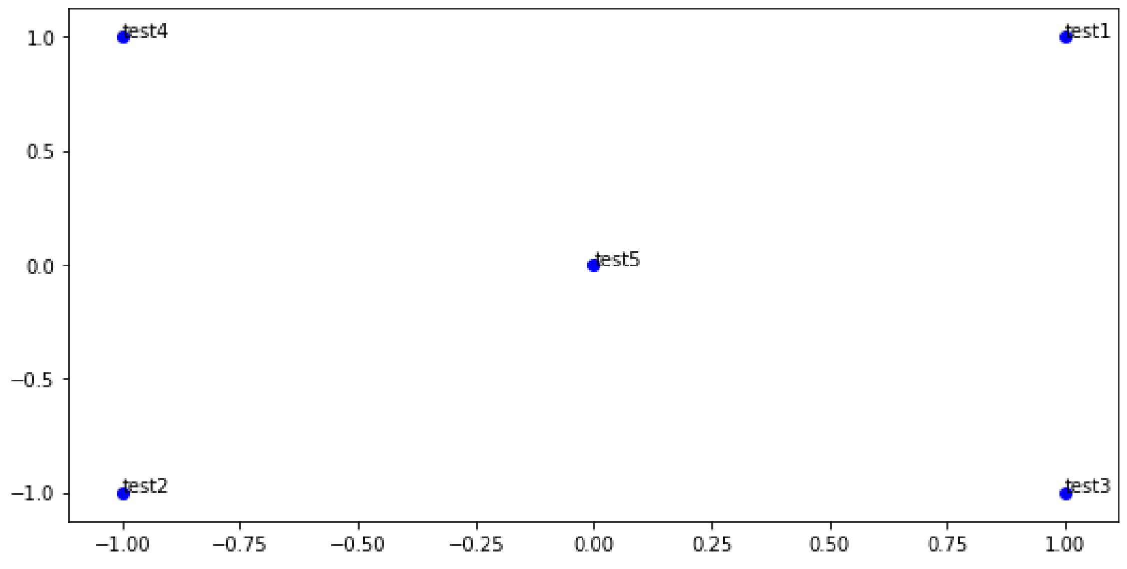
In [12]:

```
def plot_embeddings(M_reduced, word2Ind, words):
    """ Plot in a scatterplot the embeddings of the words specified in the lis
        NOTE: do not plot all the words listed in M_reduced / word2Ind.
        Include a label next to each point.

        Params:
            M_reduced (numpy matrix of shape (number of unique words in the co
            word2Ind (dict): dictionary that maps word to indices for matrix M
            words (list of strings): words whose embeddings we want to visuali
    """

    # ------------------
    # Write your implementation here.
    target_index = [word2Ind[word] for word in words]
    x_Cor = [M_reduced[x][0] for x in target_index]
    y_Cor = [M_reduced[x][1] for x in target_index]
    for k, word in enumerate(words):
        x = x_Cor[k]
        y = y_Cor[k]
        plt.scatter(x, y, marker = 'o', color = 'blue')
        plt.text(x, y, word, fontsize = 10)
    plt.show()

    # ------------------
```

In [13]:

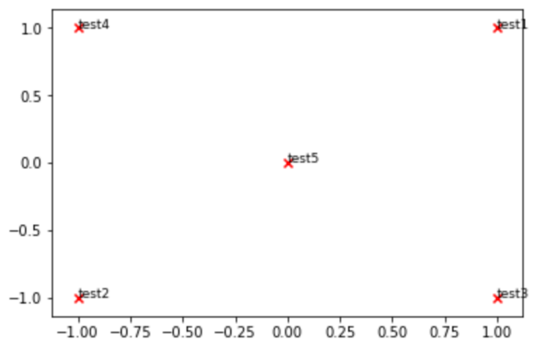
```
# ----------------------
# Run this sanity check
# Note that this not an exhaustive check for correctness.
# The plot produced should look like the "test solution plot" depicted below.
# ----------------------

print ("-" * 80)
print ("Outputted Plot:")

M_reduced_plot_test = np.array([[1, 1], [-1, -1], [1, -1], [-1, 1], [0, 0]])
word2Ind_plot_test = {'test1': 0, 'test2': 1, 'test3': 2, 'test4': 3, 'test5':
words = ['test1', 'test2', 'test3', 'test4', 'test5']
plot_embeddings(M_reduced_plot_test, word2Ind_plot_test, words)

print ("-" * 80)
```

```
----------------------------------------------------------------
-----------------
Outputted Plot:
```



```
----------------------------------------------------------------
-----------------
```

Test Plot Solution



Question 1.5: Co-Occurrence Plot Analysis [written] (10 points)

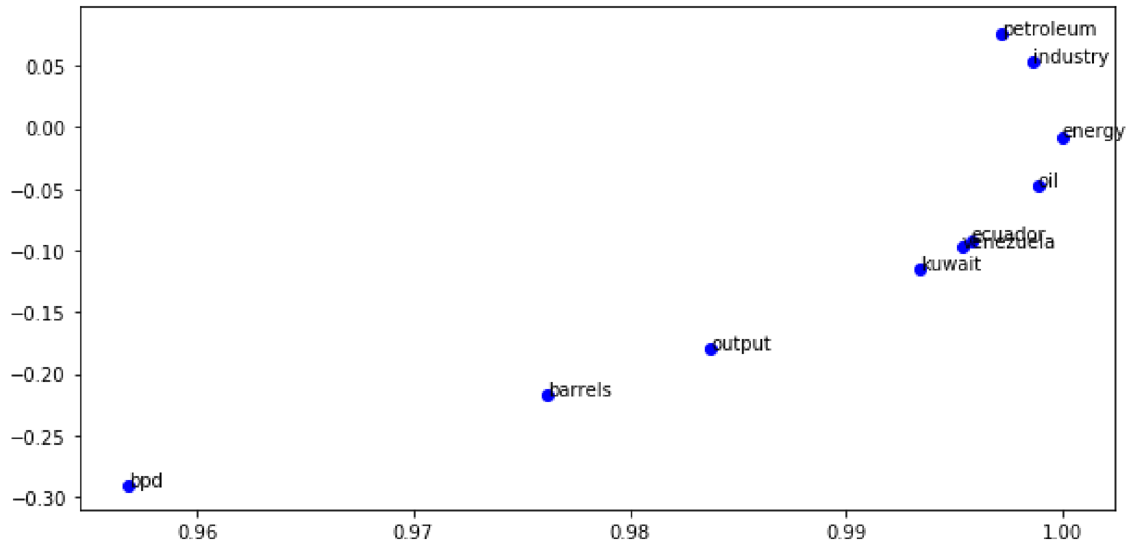
Now we will put together all the parts you have written! We will compute the co-occurrence matrix with fixed window of 4, over the Reuters "crude" corpus. Then we will use TruncatedSVD to compute 2-dimensional embeddings of each word. TruncatedSVD returns U*S, so we normalize the returned vectors, so that all the vectors will appear around the unit circle (therefore closeness is directional closeness). **Note**: The line of code below that does the normalizing uses the NumPy concept of *broadcasting*. If you don't know about broadcasting, check out Computation on Arrays: Broadcasting by Jake VanderPlas (https://jakevdp.github.io/PythonDataScienceHandbook/02.05-computation-on-arrays-broadcasting.html).

Run the below cell to produce the plot. It'll probably take a few seconds to run. What clusters together in 2-dimensional embedding space? What doesn't cluster together that you might think should have? **Note:** "bpd" stands for "barrels per day" and is a commonly used abbreviation in crude oil topic articles.

In [14]:

```
# ------------------------------
# Run This Cell to Produce Your Plot
# ------------------------------
reuters_corpus = read_corpus()
M_co_occurrence, word2Ind_co_occurrence = lab4.compute_co_occurrence_matrix(re
M_reduced_co_occurrence = lab4.reduce_to_k_dim(M_co_occurrence, k=2)

# Rescale (normalize) the rows to make them each of unit-length
M_lengths = np.linalg.norm(M_reduced_co_occurrence, axis=1)
M_normalized = M_reduced_co_occurrence / M_lengths[:, np.newaxis] # broadcast:

words = ['barrels', 'bpd', 'ecuador', 'energy', 'industry', 'kuwait', 'oil', '
plot_embeddings(M_normalized, word2Ind_co_occurrence, words)
```

```
Running Truncated SVD over 8185 words...
Done.
```



Write your answer here.

From the plot the following clusters together:

1). venezuela,kuwait,ecuador:

All top oil producing countries

2). petroleum, industry:

They may refer to petroleum industries

3). output,barrels:

Output is measure in barrels

4). oil,energy:

Oil gives energy

The following terms do not cluster together while actually these should cluster:

1). bpd and output:

Output is barrels per day

2). petroleum and oil:

Oil production can be measured as bpd

3). bpd, and barrels:

Oil industry produces some bpd amount of oil

Part 2: Prediction-Based Word Vectors (50 points)

More recently prediction-based word vectors have come into fashion, e.g. word2vec. Here, we shall explore the embeddings produced by word2vec. If you're feeling adventurous, challenge yourself and try reading the original paper (https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf).

Then run the following cells to load the word2vec vectors into memory. **Note**: This might take several minutes.

In [19]:

```
def load_word2vec():
    """ Load Word2Vec Vectors
        Return:
            wv_from_bin: All 3 million embeddings, each lengh 300
    """
    import gensim.downloader as api
    wv_from_bin = api.load("word2vec-google-news-300")
    vocab = list(wv_from_bin.index_to_key)
    print("Loaded vocab size %i" % len(vocab))
    return wv_from_bin
```

In [20]:

```
# -----------------------------------
# Run Cell to Load Word Vectors
# Note: This may take several minutes
# -----------------------------------
wv_from_bin = load_word2vec()
```

```
Loaded vocab size 3000000
```

Note: If you are receiving out of memory issues on your local machine, try closing other applications to free more memory on your device. You may want to try restarting your machine so that you can free up extra memory. Then immediately run the jupyter notebook and see if you can load the word vectors properly.

Reducing dimensionality of Word2Vec Word Embeddings

Let's directly compare the word2vec embeddings to those of the co-occurrence matrix. Run the following cells to:

1. Put the 3 million word2vec vectors into a matrix M
2. Run reduce_to_k_dim (your Truncated SVD function) to reduce the vectors from 300-dimensional to 2-dimensional.

In [23]:

```
def get_matrix_of_vectors(wv_from_bin, required_words=['barrels', 'bpd', 'ecu
    """ Put the word2vec vectors into a matrix M.
        Param:
            wv_from_bin: KeyedVectors object; the 3 million word2vec vectors l
        Return:
            M: numpy matrix shape (num words, 300) containing the vectors
            word2Ind: dictionary mapping each word to its row number in M
    """
    import random
    words = list(wv_from_bin.index_to_key)
    print("Shuffling words ...")
    random.shuffle(words)
    words = words[:10000]
    print("Putting %i words into word2Ind and matrix M..." % len(words))
    word2Ind = {}
    M = []
    curInd = 0
    for w in words:
        try:
            M.append(wv_from_bin.word_vec(w))
            word2Ind[w] = curInd
            curInd += 1
        except KeyError:
            continue
    for w in required_words:
        try:
            M.append(wv_from_bin.word_vec(w))
            word2Ind[w] = curInd
            curInd += 1
        except KeyError:
            continue
    M = np.stack(M)
    print("Done.")
    return M, word2Ind
```

In [24]:

```
# ------------------------------------------------------------------
# Run Cell to Reduce 300-Dimensinal Word Embeddings to k Dimensions
# Note: This may take several minutes
# ------------------------------------------------------------------
M, word2Ind = get_matrix_of_vectors(wv_from_bin)
M_reduced = lab4.reduce_to_k_dim(M, k=2)
```

```
Shuffling words ...
Putting 10000 words into word2Ind and matrix M...

C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.p
y:20: DeprecationWarning: Call to deprecated `word_vec` (Use ge
t_vector instead).
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.p
y:27: DeprecationWarning: Call to deprecated `word_vec` (Use ge
t_vector instead).

Done.
Running Truncated SVD over 10010 words...
Done.
```

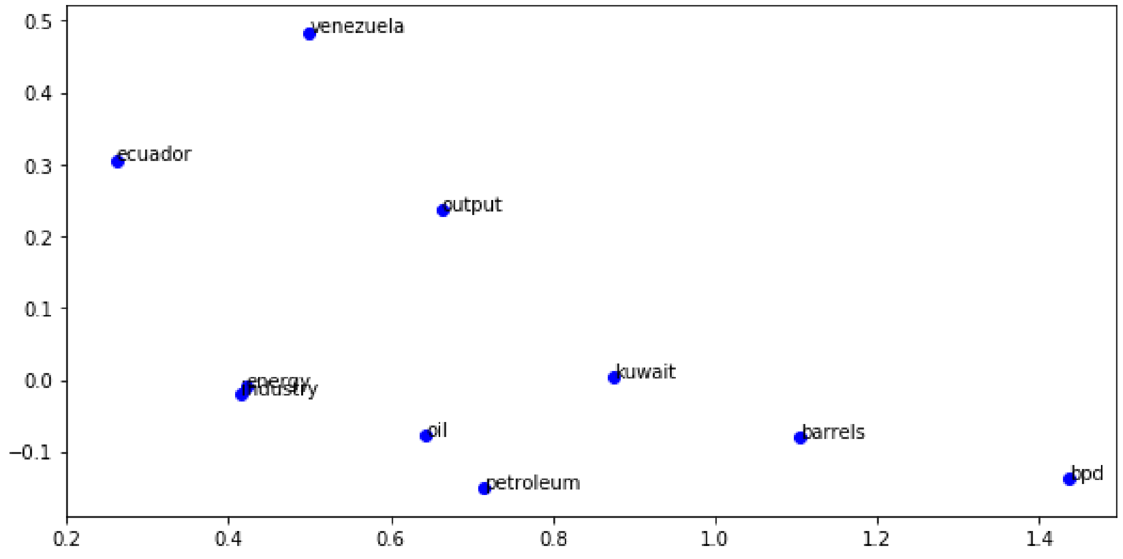
Question 2.1: Word2Vec Plot Analysis [written] (5 points)

Run the cell below to plot the 2D word2vec embeddings for `['barrels', 'bpd', 'ecuador', 'energy', 'industry', 'kuwait', 'oil', 'output', 'petroleum', 'venezuela']`.

What clusters together in 2-dimensional embedding space? What doesn't cluster together that you might think should have? How is the plot different from the one generated earlier from the co-occurrence matrix?

In [25]:

```
words = ['barrels', 'bpd', 'ecuador', 'energy', 'industry', 'kuwait', 'oil', '
plot_embeddings(M_reduced, word2Ind, words)
```



Write your answer here.

From the plot that the following items clusters together:

1). venezuela,ecuador:

All top oil producing countries but missing kuwait which is far off

2). energy, industry:

They may refer to the oil industry

3). bpd,barrels:

Bpd and barrels per day have similar meaning

4). oil,petroleum:

Oil and petroleum have the same meaning words

These terms do not cluster together:

a. bpd and output:

Output is barrels per day

b. kuwait and venezuela,ecuador:

Kuwait should be a country like others

Compared with the Word2vec plot, the co-occurence matrix plot had:

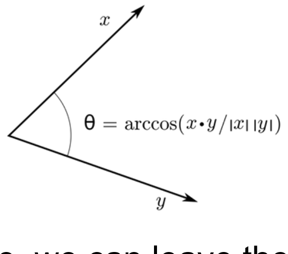
1). kuwait and venezuela,ecuador cluster together

2). barrels and output cluster together

Cosine Similarity

Now that we have word vectors, we need a way to quantify the similarity between individual words, according to these vectors. One such metric is cosine-similarity. We will be using this to find words that are "close" and "far" from one another.

We can think of n-dimensional vectors as points in n-dimensional space. If we take this perspective L1 and L2 Distances help quantify the amount of space "we must travel" to get between these two points. Another approach is to examine the angle between two vectors. From trigonometry we know that:



Instead of computing the actual angle, we can leave the similarity in terms of $similarity = cos(\Theta)$. Formally the Cosine Similarity (https://en.wikipedia.org/wiki/Cosine_similarity) s between two vectors p and q is defined as:

$$s = \frac{p \cdot q}{||p||||q||}, \text{ where } s \in [-1, 1]$$

Question 2.2: Polysemous Words (5 points) [code + written]

Find a polysemous (https://en.wikipedia.org/wiki/Polysemy) word (for example, "leaves" or "scoop") such that the top-10 most similar words (according to cosine similarity) contains related words from *both* meanings. For example, "leaves" has both "vanishes" and "stalks" in the top 10, and "scoop" has both "handed_waffle_cone" and "lowdown". You will probably need to try several polysemous words before you find one. Please state the polysemous word you discover and the multiple meanings that occur in the top 10. Why do you think many of the polysemous words you tried didn't work?

Note: You should use the `wv_from_bin.most_similar(word)` function to get the top 10 similar words. This function ranks all other words in the vocabulary with respect to their cosine similarity to the given word. For further assistance please check the **GenSim documentation (https://radimrehurek.com/gensim/models/keyedvectors.html#gensim.models.keyedve**

In [26]:

```
# ------------------
# Write your polysemous word exploration code here.

# wv_from_bin.most_similar("")

a=wv_from_bin.most_similar(['king'])
for i in a:
    print(i)

b=wv_from_bin.most_similar(['girl'])
for i in b:
    print(i)

c=wv_from_bin.most_similar(['man'])
for i in c:
    print(i)

d=wv_from_bin.most_similar(['girl'])
for i in d:
    print(i)

e=wv_from_bin.most_similar(['love'])
for i in e:
    print(i)

f=wv_from_bin.most_similar(['queen'])
for i in f:
    print(i)

g=wv_from_bin.most_similar(['king'])
for i in g:
    print(i)

h=wv_from_bin.most_similar(['woman'])
for i in h:
    print(i)
print()
# ------------------
```

```
('kings', 0.7138045430183411)
('queen', 0.6510956883430481)
('monarch', 0.6413194537162781)
('crown_prince', 0.6204220056533813)
('prince', 0.6159993410110474)
('sultan', 0.5864824056625366)
('ruler', 0.5797567367553711)
('princes', 0.5646552443504333)
('Prince_Paras', 0.5432944297790527)
('throne', 0.5422105193138123)
('boy', 0.8543271422386169)
('teenage_girl', 0.7927975654602051)
('woman', 0.7494640946388245)
('teenager', 0.7172499299049377)
('schoolgirl', 0.7075953483581543)
('teenaged_girl', 0.6650916337966919)
('daughter', 0.6489864587783813)
('mother', 0.6478164196014404)
```

```
('toddler', 0.6473966836929321)
('girls', 0.6154742240905762)
('woman', 0.7664012908935547)
('boy', 0.6824871301651001)
('teenager', 0.6586930155754089)
('teenage_girl', 0.6147903203964233)
('girl', 0.5921714305877686)
('suspected_purse_snatcher', 0.571636438369751)
('robber', 0.5585119128227234)
('Robbery_suspect', 0.5584409832954407)
('teen_ager', 0.5549196600914001)
('men', 0.5489763021469116)
('boy', 0.8543271422386169)
('teenage_girl', 0.7927975654602051)
('woman', 0.7494640946388245)
('teenager', 0.7172499299049377)
('schoolgirl', 0.7075953483581543)
('teenaged_girl', 0.6650916337966919)
('daughter', 0.6489864587783813)
('mother', 0.6478164196014404)
('toddler', 0.6473966836929321)
('girls', 0.6154742240905762)
('loved', 0.6907791495323181)
('adore', 0.6816873550415039)
('loves', 0.661863386631012)
('passion', 0.6100708842277527)
('hate', 0.600395679473877)
('loving', 0.5886635780334473)
('Ilove', 0.5702950954437256)
('affection', 0.5664337873458862)
('undying_love', 0.5547304749488831)
('absolutely_adore', 0.5536840558052063)
('queens', 0.739944338798523)
('princess', 0.7070532441139221)
('king', 0.6510956883430481)
('monarch', 0.6383602023124695)
('very_pampered_McElhatton', 0.6357026696205139)
('Queen', 0.6163407564163208)
('NYC_anglophiles_aflutter', 0.6060680150985718)
('Queen_Consort', 0.5923796892166138)
('princesses', 0.5908074975013733)
('royal', 0.5637185573577881)
('kings', 0.7138045430183411)
('queen', 0.6510956883430481)
('monarch', 0.6413194537162781)
('crown_prince', 0.6204220056533813)
('prince', 0.6159993410110474)
('sultan', 0.5864824056625366)
('ruler', 0.5797567367553711)
('princes', 0.5646552443504333)
('Prince_Paras', 0.5432944297790527)
('throne', 0.5422105193138123)
('man', 0.7664012908935547)
('girl', 0.7494640946388245)
('teenage_girl', 0.7336829304695129)
('teenager', 0.6317085027694702)
('lady', 0.6288785934448242)
('teenaged_girl', 0.6141784191131592)
('mother', 0.6076306104660034)
('policewoman', 0.6069462299346924)
('boy', 0.5975907444953918)
```

```
('Woman', 0.5770983099937439)
```

Write your answer here.

The reason for some polysemous words that didn't work might may be:

1). The corcus may not be large enought that it does not cover other words with similar meanings to that particular word.

2). The cosine similarity of these words was unable to be caught because they were mentioned in a totally different context or distance.

Question 2.3: Synonyms & Antonyms (5 points) [code + written]

When considering Cosine Similarity, it's often more convenient to think of Cosine Distance, which is simply 1 - Cosine Similarity.

Find three words (w1,w2,w3) where w1 and w2 are synonyms and w1 and w3 are antonyms, but Cosine Distance(w1,w3) < Cosine Distance(w1,w2). For example, w1="happy" is closer to w3="sad" than to w2="cheerful".

Once you have found your example, please give a possible explanation for why this counter-intuitive result may have happened.

You should use the the `wv_from_bin.distance(w1, w2)` function here in order to compute the cosine distance between two words. Please see the **GenSim documentation (https://radimrehurek.com/gensim/models/keyedvectors.html#gensim.models.keyedve** for further assistance.

In [28]:

```
# ------------------
# Write your synonym & antonym exploration code here.

w1 = "happy"
w2 = "cheerful"
w3 = "sad"
w1_w2_dist = wv_from_bin.distance(w1, w2)
w1_w3_dist = wv_from_bin.distance(w1, w3)

print("Synonyms {}, {} have cosine distance: {}".format(w1, w2, w1_w2_dist))
print("Antonyms {}, {} have cosine distance: {}".format(w1, w3, w1_w3_dist))

# ------------------
```

```
Synonyms happy, cheerful have cosine distance: 0.61622616648674
01
Antonyms happy, sad have cosine distance: 0.46453857421875
```

Write your answer here.

As we use distance based on frequency and distance/neighbourhood of words, happy & sad may occur together more often or are clustered together, which results that they have smaller cosine distance than the similar meaning words like happy & cheerful.

Solving Analogies with Word Vectors

Word2Vec vectors have been shown to *sometimes* exhibit the ability to solve analogies.

As an example, for the analogy "man : king :: woman : x", what is x?

In the cell below, we show you how to use word vectors to find x. The `most_similar` function finds words that are most similar to the words in the `positive` list and most dissimilar from the words in the `negative` list. The answer to the analogy will be the word ranked most similar (largest numerical value).

Note: Further Documentation on the `most_similar` function can be found within the **GenSim documentation (https://radimrehurek.com/gensim/models/keyedvectors.html#gensim.models.keyedve**

In [29]:

```
# Run this cell to answer the analogy -- man : king :: woman : x
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'king'], negative=['
```

```
[('queen', 0.7118193507194519),
 ('monarch', 0.6189674139022827),
 ('princess', 0.5902431011199951),
 ('crown_prince', 0.5499460697174072),
 ('prince', 0.5377321839332581),
 ('kings', 0.5236844420433044),
 ('Queen_Consort', 0.5235945582389832),
 ('queens', 0.5181134343147278),
 ('sultan', 0.5098593831062317),
 ('monarchy', 0.5087411999702454)]
```

Question 2.4: Finding Analogies [code + written] (10 Points)

Find an example of analogy that holds according to these vectors (i.e. the intended word is ranked top). In your solution please state the full analogy in the form x:y :: a:b. If you believe the analogy is complicated, explain why the analogy holds in one or two sentences.

Note: You may have to try many analogies to find one that works!

In [30]:

```
# ------------------
# Write your analogy exploration code here.

pprint.pprint(wv_from_bin.most_similar(positive=['usa','mumbai'], negative=['i

# ------------------
```

```
[('orlando', 0.5425562262535095),
 ('carlos', 0.5183202028274536),
 ('nyc', 0.5094256401062012),
 ('denver', 0.5083585977554321),
 ('montreal', 0.5047362446784973),
 ('chicago', 0.5046327710151672),
 ('miguel', 0.4954397976398468),
 ('huntington', 0.4937092661857605),
 ('jm', 0.49116042256355286),
 ('rj', 0.4910069704055786)]
```

Write your answer here.

"india : mumbai :: usa : minneaplis"

The analogy was cities in India and USA and the word vectors should predict it correctly.

Actually cities in top ten are in USA

Question 2.5: Incorrect Analogy [code + written] (5 point)

Find an example of analogy that does *not* hold according to these vectors. In your solution, state the intended analogy in the form x:y :: a:b, and state the (incorrect) value of b according to the word vectors.

In [31]:

```
# ------------------
# Write your incorrect analogy exploration code here.

pprint.pprint(wv_from_bin.most_similar(positive=['apple','blue'], negative=['r

pprint.pprint(wv_from_bin.most_similar(positive=['hand','toe'], negative=['fir

# ------------------
```

```
[('apples', 0.5500497221946716),
 ('berry', 0.5142180323600769),
 ('crumb_pie', 0.49722525477409363),
 ('peach', 0.4845479428768158),
 ('Granny_Smith', 0.4825567305088043),
 ('watermelon', 0.4808707535266876),
 ('blueberry', 0.47642412781715393),
 ('strawberry', 0.47633326053619385),
 ('Granny_Smiths', 0.473204165697097 8),
 ('pear', 0.47311925888061523)]
[('Gallinari_stubbed', 0.42742735147476196),
 ('PROBABLE_RB_Jesse_Chatman', 0.4243071973323822),
 ('hands', 0.4062863886356354),
 ('shoulder', 0.3822495937347412),
 ('toes', 0.38064050674438477),
 ('unlace', 0.37651321291923523),
 ('peep_toe_shoe', 0.3609289526939392),
 ('G_Tom_Nutten', 0.3598386347293854),
 ('hobnailed', 0.3493693470954895),
 ('polka_dot_tights', 0.34646061062812805)]
```

Write your answer here.

The "red:apple::blue:sky" and the "finger:hand::toe:foot" are predicted by this word vector collection.

The first should predict sky but it gives microsoft,ibm...

The second should predict foot but the results are shoes,hands..

Question 2.6: Guided Analysis of Bias in Word Vectors [written] (5 point)

It's important to be cognizant of the biases (gender, race, sexual orientation etc.) implicit to our word embeddings.

Run the cell below, to examine (a) which terms are most similar to "woman" and "worker" and most dissimilar to "man", and (b) which terms are most similar to "man" and "worker" and most dissimilar to "woman". What do you find in the top 10?

In [32]:

```
# Run this cell
# Here `positive` indicates the list of words to be similar to and `negative`
# most dissimilar from.
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'worker'], negative=
print()
pprint.pprint(wv_from_bin.most_similar(positive=['man', 'worker'], negative=['
```

```
[('workers', 0.6582455039024353),
 ('employee', 0.5805294513702393),
 ('nurse', 0.5249921679496765),
 ('receptionist', 0.5142489671707153),
 ('migrant_worker', 0.5001609921455383),
 ('Worker', 0.4979270100593567),
 ('housewife', 0.48609834909439087),
 ('registered_nurse', 0.484619140625),
 ('laborer', 0.48437267541885376),
 ('coworker', 0.48212406039237976)]

[('workers', 0.5590359568595886),
 ('laborer', 0.54481041431427),
 ('foreman', 0.5192232131958008),
 ('Worker', 0.5161596536636353),
 ('employee', 0.5094279646873474),
 ('electrician', 0.49481216073036194),
 ('janitor', 0.48718899488449097),
 ('bricklayer', 0.48253133893013),
 ('carpenter', 0.47499001026153564),
 ('workman', 0.46425169706344604)]
```

Write your answer here.

From the above results, it is obvious that the bias implied by word embedding is obvious, because we see that our model predicts that jobs related to women are mainly nurses, children, pregnant women and teachers, while jobs related to men are farmers, mechanics, laborers, etc.

It means bias, because men can also be teachers and nurses, and women can be technicians and laborers.

Question 2.7: Independent Analysis of Bias in Word Vectors [code + written] (10 points)

Use the `most_similar` function to find another case where some bias is exhibited by the vectors. Please briefly explain the example of bias that you discover.

In [33]:

```
# ------------------
# Write your bias exploration code here.

pprint.pprint(wv_from_bin.most_similar(positive=[ 'white','boss'], negative=['
print()
pprint.pprint(wv_from_bin.most_similar(positive=['black','boss'], negative=['w

# ------------------
```

```
[('supremo', 0.5266469717025757),
 ('head_honcho', 0.5044298768043518),
 ('bosses', 0.49901825189590454),
 ('MOTHERWELL_boss', 0.48686930537223816),
 ('honcho', 0.4774024188518524),
 ('caretaker_boss', 0.47221440076828003),
 ('WIGAN_boss', 0.4633899927139282),
 ('WEST_HAM_boss', 0.4621332585811615),
 ('manager_Stan_Ternent', 0.4565162658691406),
 ('boss_Dick_Advocaat', 0.4561653435230255)]

[('bosses', 0.571600079536438),
 ('supremo', 0.5674779415130615),
 ('chairman_Theo_Paphitis', 0.49482306838035583),
 ('Bully_Wee_boss', 0.49131691455841064),
 ('JOE_KINNEAR', 0.49093547463417053),
 ('head_honcho', 0.4908888339996338),
 ('boss_Willie_Donachie', 0.4903661906719208),
 ('WIGAN_boss', 0.48684340715408325),
 ('CARETAKER_boss', 0.4846575856208801),
 ('YEOVIL_Town_boss', 0.48320531845092773)]
```

Write your answer here.

From the results we find the race bias in the word source:

Jobs like mafia, hitman and coach relate to black race, while jobs like secretary, owner and manager relate to white race, which present race bias in the society.

Question 2.8: Thinking About Bias [written] (5 point)

What might be the cause of these biases in the word vectors?

Write your answer here.

The biases may come from the corcus material itself, because the corcus is made by some people living in the world. All people hold different degree of bias, which will affect the things they write. This implies that our model are good that it can reflect the bias inherently of the writers themselves.

In [ ]:

```

```

In [ ]:

```

```