CS224N Assignment 1

Zheng Nie

TOTAL POINTS

24 / 25

QUESTION 1

Question 1 10 pts

1.1 distinct_words [coding] 2 / 2

✓ **+ 2 pts** Correct

+ 0 pts Incorrect

1.2 compute_co_occurrence_matrix [coding] 3 / 3

✓ **- 0 pts** Correct

- 3 pts Incorrect or missing

1.3 reduce_to_k_dim [coding] 1 / 1

✓ **+ 1 pts** Passed all tests

+ 0 pts Code incorrect

1.4 plot_embeddings [coding] 1 / 1

✓ **+ 1 pts** Correct

+ 0 pts Incorrect/Unattempted

1.5 co-occurrence plot analysis [written] 3 / 3

✓ **+ 1 pts** Correct plot

✓ **+ 1 pts** Correctly noting what clusters together

✓ **+ 1 pts** Correctly noting what does not cluster together

+ 0 pts Missing/incorrect plot

QUESTION 2

Quesion 2 15 pts

2.1 word2vec plot analysis 4 / 4

✓ **+ 1 pts** Correct Plot

✓ **+ 1 pts** Clusters pointed out correctly

✓ **+ 1 pts** Non-cluster points pointed out correctly

✓ **+ 1 pts** Pointed out what's different

+ 0 pts Incorrect Plot/Unattempted

2.2 polysemous words 2 / 2

✓ **+ 1 pts** Valid polysemous word with multiple meanings in top 10

✓ **+ 1 pts** Plausible possible reason for why the chosen word's meanings show up in top 10 or why other polysemous words didn't work.

+ 0 pts No answer.

2.3 synonyms & antonyms 2 / 2

✓ **- 0 pts** Correct

- 1 pts Did not provide a possible reason

- 1 pts Did not provide valid triple

2.4 correct analogy 2 / 2

✓ **+ 2 pts** Valid analogy is top-1

+ 1 pts Valid analogy but not top-1

+ 1 pts Unclear analogy

+ 0 pts Incorrect/missing analogy

2.5 incorrect analogy 0 / 1

+ 1 pts Correct analogy, top-1 incorrect

✓ **+ 0 pts** Incorrect analogy

+ 0 pts Correct analogy, top-1 correct

+ 0 pts Correct analogy, incorrect implementation/code

+ 0 pts Not attempted

2.6 guided bias exploration 1 / 1

✓ **- 0 pts** Correct

- 1 pts Incorrect / Insufficient explanation

2.7 independent bias exploration 2 / 2

✓ **+ 1 pts** Found bias

✓ **+ 1 pts** Explained bias

+ 0 pts Incorrect/Not attempted

+ 1 pts 2-way comparison not provided and hence cannot comment on bias.

2.8 thinking about bias 1 / 1

✓ **+ 1 pts** Correct: biases are picked up from the training corpus (Google News)

+ 0 pts Insufficient/missing explanation

In [8]:
```
def distinct_words(corpus):
    """ Determine a list of distinct words for the corpus.
        Params:
            corpus (list of list of strings): corpus of documents
        Return:
            corpus_words (list of strings): list of distinct words ac
ross the corpus, sorted (using python 'sorted' function)
            num_corpus_words (integer): number of distinct words acro
ss the corpus
    """
    corpus_words = []
    num_corpus_words = -1

    # ------------------
    # Write your implementation here.
    corpus_words = sorted(list(set(j for i in corpus for j in i)))
    num_corpus_words = len(corpus_words)

    # ------------------
    return corpus_words, num_corpus_words
```

In [9]:
```
# ---------------------
# 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's well that ends well END".split(" ")]
test_corpus_words, num_corpus_words = distinct_words(test_corpus)

# Correct answers
ans_test_corpus_words = sorted(list(set(["START", "All", "ends", "tha
t", "gold", "All's", "glitters", "isn't", "well", "END"])))
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 distinct words. Correct: {}. Yours: {}".format(ans_num_corpus_wor
ds, num_corpus_words)

# Test correct words
assert (test_corpus_words == ans_test_corpus_words), "Incorrect corpu
s_words.\nCorrect: {}\nYours:   {}".format(str(ans_test_corpus_words
), str(test_corpus_words))

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

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

Question 1.2: Implement `compute_co_occurrence_matrix` [code] (3 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. If you're not familiar with NumPy, there's a NumPy tutorial in the second half of this cs231n Python NumPy tutorial (http://cs231n.github.io/python-numpy-tutorial/).

1.1 distinct_words [coding] **2 / 2**

✓ **+ 2 pts** **Correct**

+ 0 pts Incorrect

In [10]:
```
def compute_co_occurrence_matrix(corpus, window_size=4):
    """ Compute co-occurrence matrix for the given corpus and window_
size (default of 4).

        Note: Each word in a document should be at the center of a wi
ndow. Words near edges will have a smaller
              number of co-occurring words.

              For example, if we take the document "START All that gl
itters is not gold END" with window size of 4,
              "All" will co-occur with "START", "that", "glitters",
 "is", and "not".

        Params:
            corpus (list of list of strings): corpus of documents
            window_size (int): size of context window
        Return:
            M (numpy matrix of shape (number of corpus words, number
 of number of corpus words)):
                Co-occurence matrix of word counts.
                The ordering of the words in the rows/columns should
 be the same as the ordering of the words given by the distinct_words
 function.
            word2Ind (dict): dictionary that maps word to index (i.e.
 row/column number) for matrix M.
    """
    words, num_words = distinct_words(corpus)
    M = None

    word2Ind = {}

    # ------------------
    # Write your implementation here.
    M = np.zeros((num_words, num_words))

    # generate word2Ind for M indexing
    for i, word in enumerate(words):
        word2Ind[word] = i

    for corp in corpus:
        for s,sub_string in enumerate(corp):
            sub_string_word_to_check = corp[0 if s-window_size < 0 el
se s-window_size :s]
            sub_string_word_to_check.extend(corp[s+1:s+window_size+1
])
            i = word2Ind[sub_string]
            for string in sub_string_word_to_check:
                M[i][word2Ind[string]] = M[i][word2Ind[string]] + 1.0

    # ------------------

    return M, word2Ind
```

In [11]:
```
# ---------------------
# 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's well that ends well END".split(" ")]
M_test, word2Ind_test = compute_co_occurrence_matrix(test_corpus, win
dow_size=1)

# 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
, 'glitters': 5, 'gold': 6, "isn't": 7, 'that': 8, 'well': 9}

# Test correct word2Ind
assert (word2Ind_ans == word2Ind_test), "Your word2Ind is incorrect:
\nCorrect: {}\nYours: {}".format(word2Ind_ans, word2Ind_test)

# Test correct M shape
assert (M_test.shape == M_test_ans.shape), "M matrix has incorrect sh
ape.\nCorrect: {}\nYours: {}".format(M_test.shape, M_test_ans.shape)

# 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 ({}, {})=(
{}, {}) in matrix M. Yours has {} but should have {}.".format(idx1, i
dx2, w1, w2, student, correct))

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

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

Question 1.3: Implement `reduce_to_k_dim` [code] (1 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 [12]:
```
def reduce_to_k_dim(M, k=2):
    """ Reduce a co-occurence count matrix of dimensionality (num_cor
pus_words, num_corpus_words)
        to a matrix of dimensionality (num_corpus_words, k) using the
 following SVD function from Scikit-Learn:
            - http://scikit-learn.org/stable/modules/generated/sklear
n.decomposition.TruncatedSVD.html

        Params:
            M (numpy matrix of shape (number of corpus words, number
 of number of corpus words)): co-occurence matrix of word counts
            k (int): embedding size of each word after dimension redu
ction
        Return:
            M_reduced (numpy matrix of shape (number of corpus words,
 k)): matrix of k-dimensioal word embeddings.
                    In terms of the SVD from math class, this actuall
y returns U * S
    """
    n_iters = 10     # Use this parameter in your call to `TruncatedS
VD`
    M_reduced = None
    print("Running Truncated SVD over %i words..." % (M.shape[0]))

    # ------------------
    # Write your implementation here.
    svd = TruncatedSVD(n_components=k, n_iter=n_iters)
    M_reduced = svd.fit_transform(M)
    # ------------------

    print("Done.")
    return M_reduced
```

1.2 compute_co_occurrence_matrix [coding] 3 / 3

✓ - 0 pts Correct

- 3 pts Incorrect or missing

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

Question 1.3: Implement `reduce_to_k_dim` [code] (1 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 [12]:
```
def reduce_to_k_dim(M, k=2):
    """ Reduce a co-occurence count matrix of dimensionality (num_cor
pus_words, num_corpus_words)
        to a matrix of dimensionality (num_corpus_words, k) using the
 following SVD function from Scikit-Learn:
            - http://scikit-learn.org/stable/modules/generated/sklear
n.decomposition.TruncatedSVD.html

        Params:
            M (numpy matrix of shape (number of corpus words, number
 of number of corpus words)): co-occurence matrix of word counts
            k (int): embedding size of each word after dimension redu
ction
        Return:
            M_reduced (numpy matrix of shape (number of corpus words,
 k)): matrix of k-dimensioal word embeddings.
                    In terms of the SVD from math class, this actuall
y returns U * S
    """
    n_iters = 10     # Use this parameter in your call to `TruncatedS
VD`
    M_reduced = None
    print("Running Truncated SVD over %i words..." % (M.shape[0]))

    # ------------------
    # Write your implementation here.
    svd = TruncatedSVD(n_components=k, n_iter=n_iters)
    M_reduced = svd.fit_transform(M)
    # ------------------

    print("Done.")
    return M_reduced
```

In [13]:
```
# ---------------------
# 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's well that ends well END".split(" ")]
M_test, word2Ind_test = compute_co_occurrence_matrix(test_corpus, win
dow_size=1)
M_test_reduced = reduce_to_k_dim(M_test, k=2)

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

# 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] (1 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.

1.3 reduce_to_k_dim [coding] 1 / 1

✓ **+ 1 pts** **Passed all tests**

+ 0 pts Code incorrect

In [14]:
```
def plot_embeddings(M_reduced, word2Ind, words):
    """ Plot in a scatterplot the embeddings of the words specified i
n the list "words".
        NOTE: do not plot all the words listed in M_reduced / word2In
d.
        Include a label next to each point.

        Params:
            M_reduced (numpy matrix of shape (number of corpus words,
 k)): matrix of k-dimensioal word embeddings
            word2Ind (dict): dictionary that maps word to indices for
 matrix M
            words (list of strings): words whose embeddings we want t
o visualize
    """

    # ------------------
    # Write your implementation here.
    for word in words:
        coords = M_reduced[word2Ind[word]]
        x = coords[0]
        y = coords[1]
        plt.scatter(x, y, marker='x', color='red')
        plt.text(x, y, word, fontsize=9)
    plt.show()

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

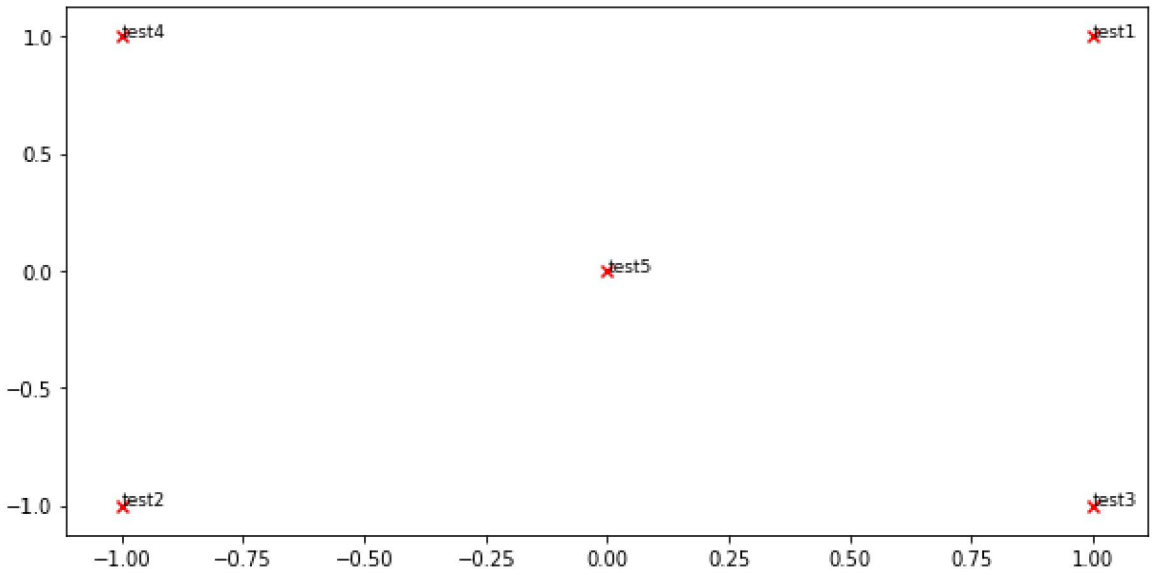
In [15]:
```
# ---------------------
# Run this sanity check
# Note that this not an exhaustive check for correctness.
# The plot produced should look like the "test solution plot" depicte
d below.
# ---------------------

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': 4}
words = ['test1', 'test2', 'test3', 'test4', 'test5']
plot_embeddings(M_reduced_plot_test, word2Ind_plot_test, words)

print ("-" * 80)
```

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




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

1.4 plot_embeddings [coding] 1 / 1

✓ **+ 1 pts** **Correct**

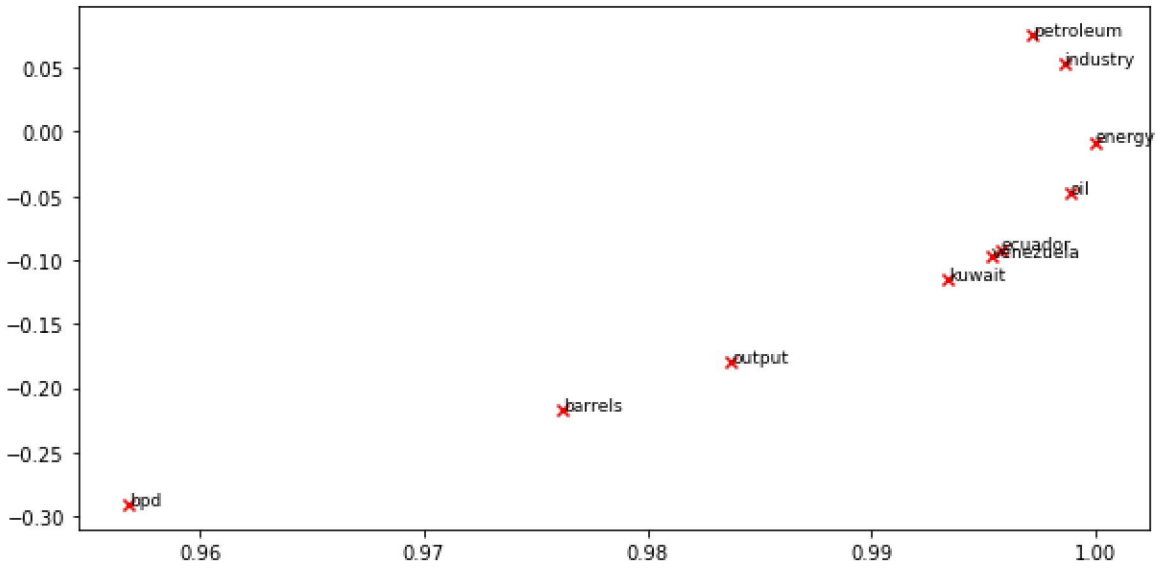
+ 0 pts Incorrect/Unattempted

In [16]:
```
# ----------------------------
# Run This Cell to Produce Your Plot
# ----------------------------
reuters_corpus = read_corpus()
M_co_occurrence, word2Ind_co_occurrence = compute_co_occurrence_matri
x(reuters_corpus)
M_reduced_co_occurrence = 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] # b
roadcasting

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

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



Answer:

kuwait,venezuela,ecuador which are all countries (names) have clustered together. Those countries also close to oil, since those country produces a lot of oil. energy and oil are closed since oil can generate genergy so it is related to energy.Petroleum and industry are closed since petroleum is kind of industry,so industry is generalizion of petroleum.

oil and petroleum should clustered, since they have similar meaning. bpd and output should have clustered as they could have samilar meaning of the productitity outputs of petroleum industry

1.5 co-occurrence plot analysis [written] 3 / 3

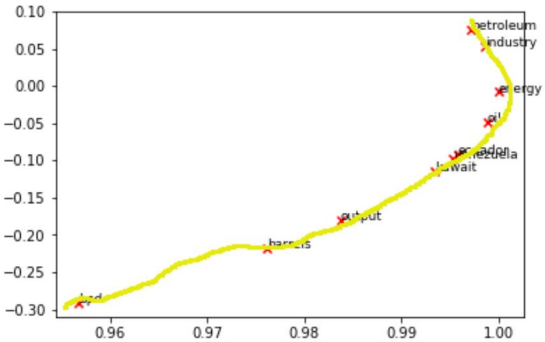
- ✓ + 1 pts Correct plot
- ✓ + 1 pts Correctly noting what clusters together
- ✓ + 1 pts Correctly noting what does not cluster together
- + 0 pts Missing/incorrect plot

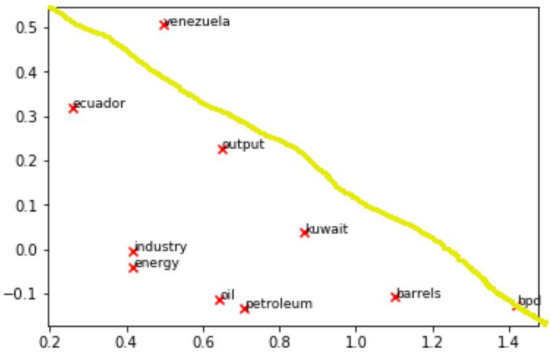
Answer:

oil and petroleum which have similar meaning are clustered. industry and energy are clustered.<

ecuador, kuwait, venezuela whitch are all petroleum production country name should have clustered,output and bpd have samilar meaning for petroleum production amount should have clustered

The difference this plot from the one generated eariler is: First, the scale of X, Y asixes is different. second, the distribution in plot for same word is different between those 2 plot, in the plot generate eariler we could see all the words distrion is along/closed to a curved line, for the second one we just generated is spread in the left down side of the plot(See following two figures). Third different is the evaluation result is different. in the plot generated earilier shows those oil contries names are clustered together and close to oil. in the new one, we didn`t see this. Forth, for same word in those 2 word embedding plot the word vector weight/value (coordinates) is different. For example, for oil in first plot, it is (1.00, -0.05), the oil in second plot has (0.62, -0.11)

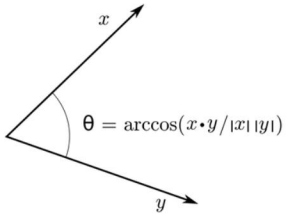




Cosine Similarity

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:

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

2.1 word2vec plot analysis **4 / 4**

- ✓ **+ 1 pts** **Correct Plot**
- ✓ **+ 1 pts** **Clusters pointed out correctly**
- ✓ **+ 1 pts** **Non-cluster points pointed out correctly**
- ✓ **+ 1 pts** **Pointed out what's different**
- **+ 0 pts** Incorrect Plot/Unattempted

Question 2.2: Polysemous Words (2 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.keyedvectors.FastTextKeye**

In [23]:
```
# ------------------
# Write your polysemous word exploration code here.

wv_from_bin.most_similar("left")

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

Out[23]:
```
[('leaving', 0.6707000732421875),
 ('leave', 0.525093138217926),
 ('leaves', 0.5228644609451294),
 ('returned', 0.5059226751327515),
 ('right', 0.49213993549346924),
 ('departed', 0.49109700322151184),
 ('limped', 0.48599502444267273),
 ('went', 0.4719873070716858),
 ('remaining', 0.4650370478630066),
 ('empty', 0.4546155333518982)]
```

Answer:

The word I choose is "left", which has "leave" and "remaining".

Most of the meanings of the polysemous have are unrelated, it is possible that the meaning cluster for those meanings in the embedding spaces are not close to each other. and the number of similar words for each cluster could have more than the polysemous words and more closer to each other in word embedding spaces.which also means the one or some polysemous in the corpus traing the wv are more often used or "contexted" than others. and also only pick top 10, so it even reduces the chance to show the simility for polysemous in the list.

2.2 polysemous words 2 / 2

✓ + 1 pts Valid polysemous word with multiple meanings in top 10

✓ + 1 pts Plausible possible reason for why the chosen word's meanings show up in top 10 or why other polysemous words didn't work.

+ 0 pts No answer.

Question 2.3: Synonyms & Antonyms (2 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.keyedvectors.FastTextKeye** for further assistance.

In [24]:
```
# ------------------
# Write your synonym & antonym exploration code here.

w1 = "large"
w2 = "gaint"
w3 = "small"
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 large, gaint have cosine distance: 0.8168608269597246
Antonyms large, small have cosine distance: 0.26688501049911195
```

Answer

The word I found is large, gaint, small.

Here`s the reason why this conter-intuitive result may have happened:
if we run wv_from_bin.most_similar("large") like question 2.2, we will get:

[('sizeable', 0.7341436147689819),
('small', 0.733115017414093),
('sizable', 0.7325241565704346),
('Large', 0.6654301285743713),
('huge', 0.6589166522026062),
('larger', 0.6517370939254761),
('massive', 0.6127927899360657),
('smaller', 0.6025481224060059),
('substantial', 0.5873902440071106),
('gigantic', 0.5858762860298157)] ,

if we run wv_from_bin.most_similar("small") we will get

[('large', 0.733115017414093),
('tiny', 0.718792736530304),
('medium_sized', 0.6426598429679871),
('Small', 0.6270469427108765),
('smaller', 0.619428813457489),
('minuscule', 0.5676835775375366),
('larger', 0.5446805953979492),
('mid_sized', 0.5328439474105835),
(midsized', 0.519831120967865),
('sizable', 0.5175076723098755)],

if we run wv_from_bin.most_similar("gaint"), we will get

[('giant', 0.5869466066360474),
('BOMBAY_XFN_ASIA', 0.4832102656364441),
('jt_venture', 0.4771811068058014),
('Videocon_Industries_Ltd.', 0.4737812578678131)
('TPV_Technology_Ltd', 0.46327781677246094),
('outsourcer_Wipro', 0.45941978693008423),
('RELI.BO', 0.45818576216697693),
('Ltd._Indiaā', 0.4561852216720581),
('Jindal_Saw_Ltd', 0.4555988311767578),
('retailer_Gome', 0.45485517382621765)]

For result above , we could see for large and small, the top 10 words are almost same and most focusing on the size property description. for gaint, which is also size property desction word, expecting showing similar result, but top 10 is showing most of them are company names. which means in the word embedding space, word

vector for word "gaint" are closer to the company name clusters. and word vector for "large" and "small" are closer to size descriptions clusters. which also means in the corpus for traing w2v , the gaint are most often being used for comanies or it self represents big companines rather than being used for size.

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.keyedvectors.FastTextKeye**

In [25]:
```
# Run this cell to answer the analogy -- man : king :: woman : x
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'king'], ne
gative=['man']))
```

```
[('queen', 0.7118192315101624),
 ('monarch', 0.6189675331115723),
 ('princess', 0.5902431011199951),
 ('crown_prince', 0.5499460697174072),
 ('prince', 0.5377321243286133),
 ('kings', 0.5236844420433044),
 ('Queen_Consort', 0.5235946178436279),
 ('queens', 0.5181134343147278),
 ('sultan', 0.5098592638969421),
 ('monarchy', 0.5087411999702454)]
```

Question 2.4: Finding Analogies [code + written] (2 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!

2.3 synonyms & antonyms **2 / 2**

✓ **- 0 pts** **Correct**

- 1 pts Did not provide a possible reason

- 1 pts Did not provide valid triple

In [26]:
```
# ------------------
# Write your analogy exploration code here.

pprint.pprint(wv_from_bin.most_similar(positive=['man', 'actress'], n
egative=['woman']))
print()
pprint.pprint(wv_from_bin.most_similar(positive=['man', 'girl'], nega
tive=['woman']))
print()
pprint.pprint(wv_from_bin.most_similar(positive=['large', 'low'], neg
ative=['small']))


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

```
[('actor', 0.8202018141746521),
 ('Actor', 0.6390689611434937),
 ('funnyman', 0.6047691106796265),
 ('thespian', 0.5847228765487671),
 ('thesp', 0.5750559568405151),
 ('heart_throb', 0.5708590745925903),
 ('starlet', 0.5672707557678223),
 ('singer', 0.5657444000244141),
 ('Actress', 0.5615350008010864),
 ('ADRIEN_BRODY', 0.5556015372276306)]

[('boy', 0.8748372793197632),
 ('teenager', 0.6932151913642883),
 ('teenage_girl', 0.6277071237564087),
 ('lad', 0.624101996421814),
 ('kid', 0.6167631149291992),
 ('schoolboy', 0.600092351436615),
 ('teen_ager', 0.5852149724960327),
 ('boys', 0.5807873606681824),
 ('youngster', 0.5773991346359253),
 ('guy', 0.5459756255149841)]

[('high', 0.6629930734634399),
 ('Low', 0.5137864947319031),
 ('lower', 0.5012373924255371),
 ('higher', 0.49018242955207825),
 ('lowest', 0.48971161246299744),
 ('abysmally_low', 0.4538779556751251),
 ('highest', 0.4514465928077698),
 ('levels', 0.4280760586261749),
 ('rising', 0.4265591502189636),
 ('ahigh', 0.4251363277435303)]
```

Answer:

woman:actress::man:actor

woman:girl::man:boy

small:low::large:high

Question 2.5: Incorrect Analogy [code + written] (1 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 [27]:
```
# ------------------
# Write your incorrect analogy exploration code here.

pprint.pprint(wv_from_bin.most_similar(positive=['woman','doctor'], n
egative=['man']))

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

```
[('gynecologist', 0.7093890905380249),
 ('nurse', 0.6477286219596863),
 ('doctors', 0.6471461653709412),
 ('physician', 0.64389967918396),
 ('pediatrician', 0.6249487400054932),
 ('nurse_practitioner', 0.6218313574790955),
 ('obstetrician', 0.6072014570236206),
 ('ob_gyn', 0.598671317100524 9),
 ('midwife', 0.5927063226699829),
 ('dermatologist', 0.5739566683769226)]
```

Answer:

Intended: man:doctor::woman:physician, actual: man:doctor::woman:gynecologist, gynecologist should be more generalized.

Question 2.6: Guided Analysis of Bias in Word Vectors [written] (1 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 "boss" and most dissimilar to "man", and (b) which terms are most similar to "man" and "boss" and most dissimilar to "woman". What do you find in the top 10?

2.4 correct analogy 2 / 2

- ✓ **+ 2 pts** **Valid analogy is top-1**
- **+ 1 pts** Valid analogy but not top-1
- **+ 1 pts** Unclear analogy
- **+ 0 pts** Incorrect/missing analogy

Answer:

woman:actress::man:actor

woman:girl::man:boy

small:low::large:high

Question 2.5: Incorrect Analogy [code + written] (1 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 [27]:
```
# ------------------
# Write your incorrect analogy exploration code here.

pprint.pprint(wv_from_bin.most_similar(positive=['woman','doctor'], n
egative=['man']))

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

```
[('gynecologist', 0.7093890905380249),
 ('nurse', 0.6477286219596863),
 ('doctors', 0.6471461653709412),
 ('physician', 0.64389967918396),
 ('pediatrician', 0.6249487400054932),
 ('nurse_practitioner', 0.6218313574790955),
 ('obstetrician', 0.6072014570236206),
 ('ob_gyn', 0.598671317100524 9),
 ('midwife', 0.5927063226699829),
 ('dermatologist', 0.5739566683769226)]
```

Answer:

Intended: man:doctor::woman:physician, actual: man:doctor::woman:gynecologist, gynecologist should be more generalized.

Question 2.6: Guided Analysis of Bias in Word Vectors [written] (1 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 "boss" and most dissimilar to "man", and (b) which terms are most similar to "man" and "boss" and most dissimilar to "woman". What do you find in the top 10?

2.5 incorrect analogy **0 / 1**

+ **1 pts** Correct analogy, top-1 incorrect

✓ **+ 0 pts** **Incorrect analogy**

+ **0 pts** Correct analogy, top-1 correct

+ **0 pts** Correct analogy, incorrect implementation/code

+ **0 pts** Not attempted

In [30]:
```
# Run this cell
# Here `positive` indicates the list of words to be similar to and `n
egative` indicates the list of words to be
# most dissimilar from.
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'boss'], ne
gative=['man']))
print()
pprint.pprint(wv_from_bin.most_similar(positive=['man', 'boss'], nega
tive=['woman']))
```

```
[('bosses', 0.552264392375946),
 ('manageress', 0.49151355028152466),
 ('exec', 0.45940810441970825),
 ('Manageress', 0.4559843242168426 5),
 ('receptionist', 0.4474116563796997),
 ('Jane_Danson', 0.44480547308921814),
 ('Fiz_Jennie_McAlpine', 0.44275766611099243),
 ('Coronation_Street_actress', 0.44275569915771484),
 ('supremo', 0.4409852623939514),
 ('coworker', 0.4398624897003174)]

[('supremo', 0.6097398400306702),
 ('MOTHERWELL_boss', 0.5489562153816223),
 ('CARETAKER_boss', 0.5375303626060486),
 ('Bully_Wee_boss', 0.533397376537323),
 ('YEOVIL_Town_boss', 0.5321705341339111),
 ('head_honcho', 0.5281979441642761),
 ('manager_Stan_Ternent', 0.5259714722633362),
 ('Viv_Busby', 0.5256163477897644),
 ('striker_Gabby_Agbonlahor', 0.5250812768936157),
 ('BARNSLEY_boss', 0.5238943696022034)]
```

Answer:

The prediction of positive=['woman', 'boss'], negative=['man']) is less appropriate than positive=['man', 'boss'], negative=['woman'], "receptionist", "Coronation_Stree_actree", "coworker" are inappropriate even disrespectful like cornation_stree_actress, bosses is complex of boss, however man and woman are all singular

Question 2.7: Independent Analysis of Bias in Word Vectors [code + written] (2 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.

2.6 guided bias exploration 1 / 1

✓ **- 0 pts** **Correct**

- 1 pts Incorrect / Insufficient explanation

In [29]:
```
# ------------------
# Write your bias exploration code here.

pprint.pprint(wv_from_bin.most_similar(positive=['woman','doctor'], n
egative=['man']))
print()
pprint.pprint(wv_from_bin.most_similar(positive=['man','doctor'], neg
ative=['woman']))

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

```
[('gynecologist', 0.7093890905380249),
 ('nurse', 0.6477286219596863),
 ('doctors', 0.6471461653709412),
 ('physician', 0.64389967918396),
 ('pediatrician', 0.6249487400054932),
 ('nurse_practitioner', 0.6218313574790955),
 ('obstetrician', 0.6072014570236206),
 ('ob_gyn', 0.5986713171005249),
 ('midwife', 0.5927063226699829),
 ('dermatologist', 0.5739566683769226)]

[('physician', 0.646366536617279),
 ('doctors', 0.5858404636383057),
 ('surgeon', 0.572394073009491),
 ('dentist', 0.552364706993103),
 ('cardiologist', 0.5413815975189209),
 ('neurologist', 0.5271126627922058),
 ('neurosurgeon', 0.5249835848808289),
 ('urologist', 0.5247740149497986),
 ('Doctor', 0.5240625143051147),
 ('internist', 0.5183224081993103)]
```

Answer:

The prediction for positive=['man','doctor'], negative=['woman'] is much more accure than positive=['woman','doctor'], negative=['man'], we expect man:doctor:woman:physician for both two above. Woman and man should have equal opportunity of being doctor/physician rather than just gynecologist at the most, which is not equal for woman and inappropriate, it`s same problem as shown in question 2.6

Question 2.8: Thinking About Bias [written] (1 point)

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

2.7 independent bias exploration **2 / 2**

✓ **+ 1 pts** **Found bias**

✓ **+ 1 pts** **Explained bias**

+ 0 pts Incorrect/Not attempted

+ 1 pts 2-way comparison not provided and hence cannot comment on bias.

Answer:

The cause of the biases may come from the corpus for training the word vector, some words have less context content existance consistent with the common human sense(correct answer) in the corpus，so it is similar like when training a machine leaning model with a small number of dataset may have bias. For example, in question Question 2.7, man:doctor::woman:gynecologist, however it should be physician as man:doctor::woman:physician, it may because woman have most context with gynecologist in the corpus may due to most of the OB doctor is woman. so the wv for gynecologist is closer than physician for woman

Submission Instructions

1. Click the Save button at the top of the Jupyter Notebook.
2. Please make a Gradescope account using your @stanford.edu email address (this is very important to help us enter your grade at the end of the quarter), and ensure your SUID is entered at the top of this notebook too.
3. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of all cells).
4. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.
5. Once you've rerun everything, select File -> Download as -> PDF via LaTeX
6. Look at the PDF file and make sure all your solutions are there, displayed correctly. The PDF is the only thing your graders will see!
7. Submit your PDF on Gradescope.

In [ ]:

2.8 thinking about bias 1 / 1

✓ **+ 1 pts** **Correct: biases are picked up from the training corpus (Google News)**

+ 0 pts Insufficient/missing explanation