CS 539: Natural Language Processing with Deep Learning **(Due: January 20, 2021)**

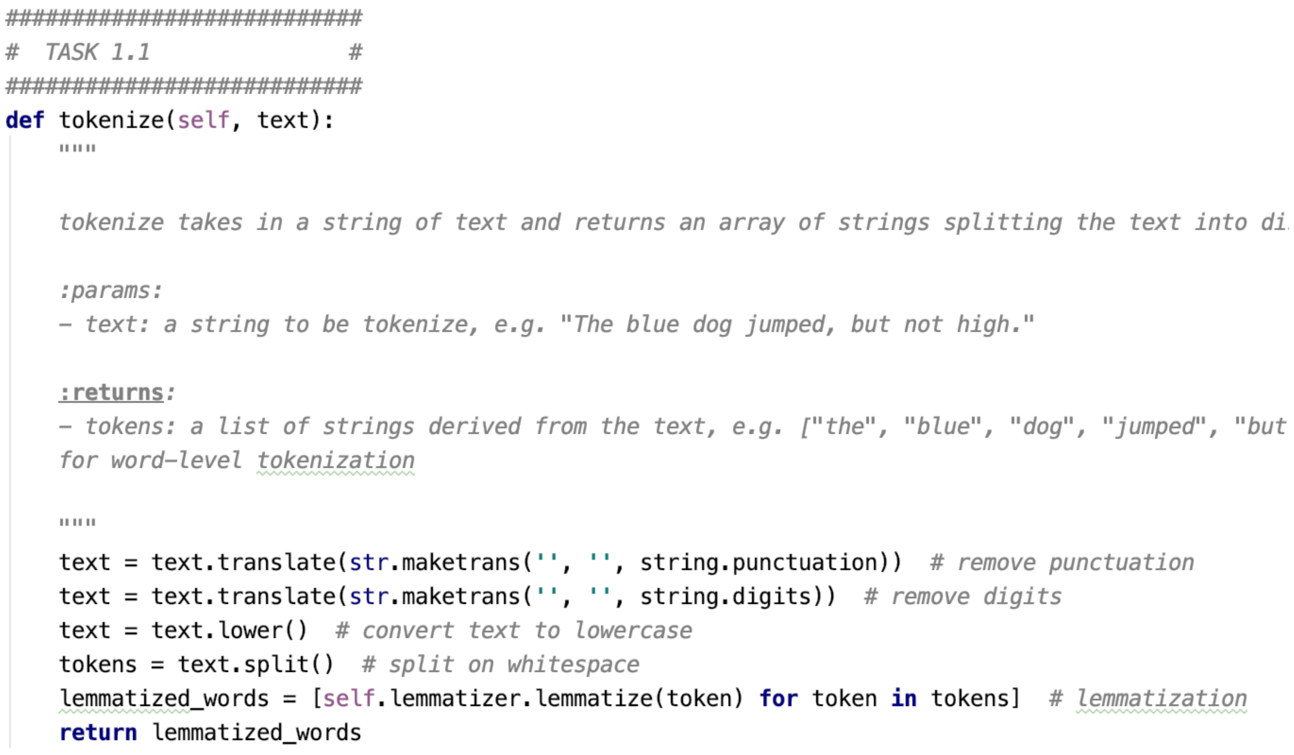
Submission Homework 1

Instructor: Prof. Stefan Lee *Name:* Rashmi Jadhav, *Student ID:* 934-069-574

1. Getting Used to Handling Language Data

1. All the punctuations and digits were first removed from the input text/sentence and then the text was converted to lowercase so that "Sport" and "sport" would be treated the same; this also reduces the number of tokens we'd eventually produce. The processed text was split on whitespaces to convert to a list of tokens. NLTK's word_tokenizer() and a few more tokenizers were tried out but they handled contractions in a different way (splitting **"can't"** to **"ca", "'nt"**) and these are probably eliminated later by stopwords removal but sticking to traditional python way of tokenizing was chosen as it was faster(although meaning-wise worse approach than that of word tokenizer).

   Lemmatization was tried using NLTK's WordNetLemmatizer(). This is different than stemming in the sense that it doesn't just try to chop off the end of a word but tries to do it meaningfully using vocabulary. Again, this lemmatizer didn't do that well by passing just the word since it treats every word as noun and tries to find this noun's lemma. For best performance of the lemmatizer, each token was lemmatized along with its POS tag. However, using NLTK's **pos_tag()** gave a significant performance hit causing the processing time to increase from about 10s to 10mins. Thus, the processed text was lemmatized without POS tag.

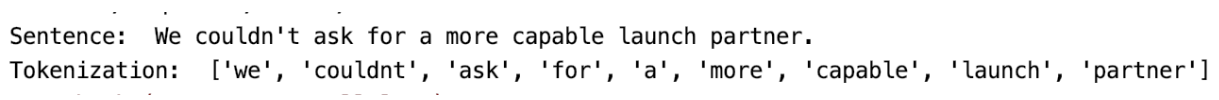


```
##########################
#  TASK 1.1              #
##########################
def tokenize(self, text):
    """

    tokenize takes in a string of text and returns an array of strings splitting the text into di

    :params:
    - text: a string to be tokenize, e.g. "The blue dog jumped, but not high."

    :returns:
    - tokens: a list of strings derived from the text, e.g. ["the", "blue", "dog", "jumped", "but
    for word-level tokenization

    """
    text = text.translate(str.maketrans('', '', string.punctuation))  # remove punctuation
    text = text.translate(str.maketrans('', '', string.digits))  # remove digits
    text = text.lower()  # convert text to lowercase
    tokens = text.split()  # split on whitespace
    lemmatized_words = [self.lemmatizer.lemmatize(token) for token in tokens]  # lemmatization
    return lemmatized_words
```

Following is an example input and tokenized output:

```
Sentence:  With Funding From Jeff Bezos,
 MethaneSAT Picks Elon Musk's SpaceX for 2022 Launch.
Tokenization:  ['with', 'funding', 'from', 'jeff', 'bezos', 'methanesat', 'pick', 'elon',
  'musk', 'spacex', 'for', 'launch']
```

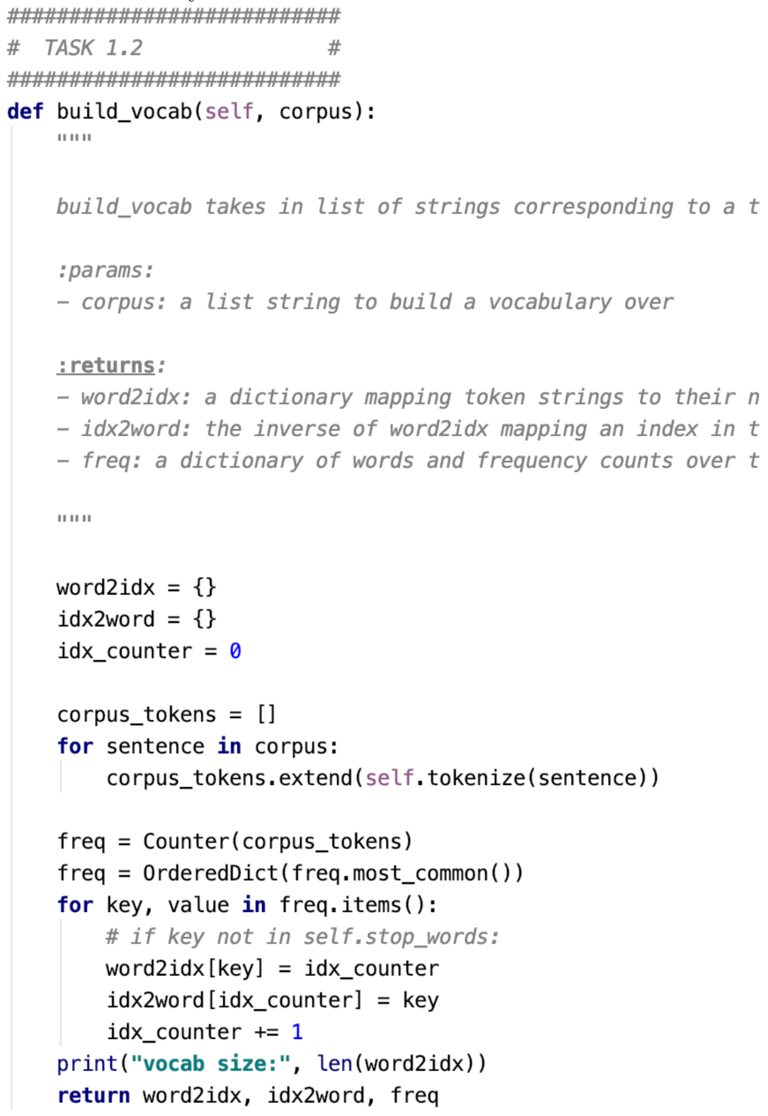
Notice how '\n' has been removed, non-alphabetic 2022 is removed, picks is lemmatized to pick, funding here is a noun and hence it is not lemmatized to 'fund'.



```
Sentence:  We couldn't ask for a more capable launch partner.
Tokenization:  ['we', 'couldnt', 'ask', 'for', 'a', 'more', 'capable', 'launch', 'partner']
```

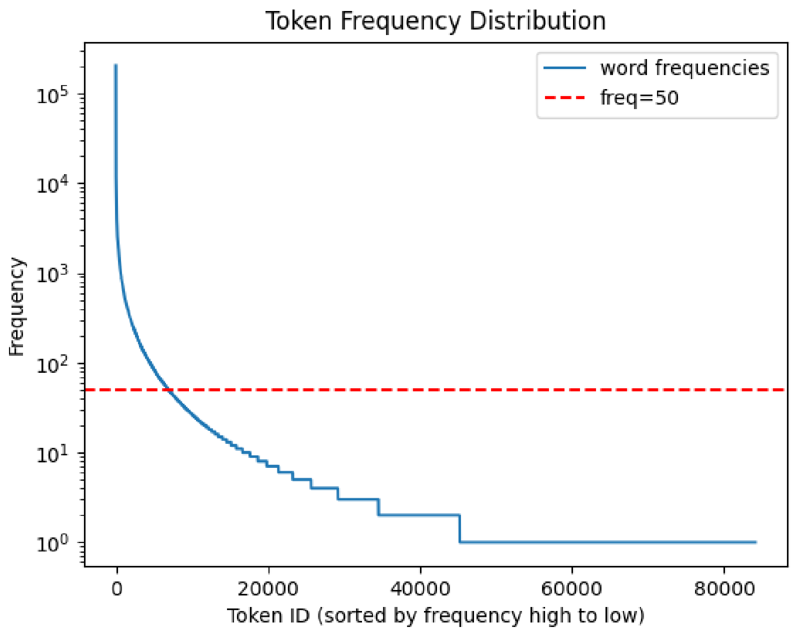
Notice how contraction **couldn't** is preserved. Although it might be a word that doesn't contribute much to the sentence meaning.
All in all, a text may contain a lot of different kinds of words like misspelled words, http urls, and other unexpected tokens and thus tokenizing seemed to be quite a difficult task.

2. Build vocabulary and fill in the vocab data structure:

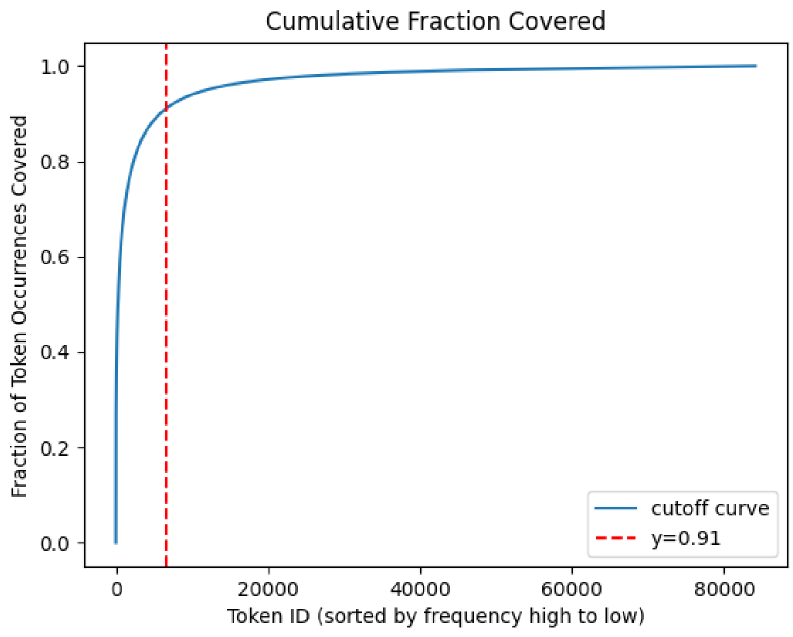


```
###########################
#  TASK 1.2               #
###########################
def build_vocab(self, corpus):
    """

    build_vocab takes in list of strings corresponding to a t

    :params:
    - corpus: a list string to build a vocabulary over

    :returns:
    - word2idx: a dictionary mapping token strings to their n
    - idx2word: the inverse of word2idx mapping an index in t
    - freq: a dictionary of words and frequency counts over t

    """

    word2idx = {}
    idx2word = {}
    idx_counter = 0

    corpus_tokens = []
    for sentence in corpus:
        corpus_tokens.extend(self.tokenize(sentence))

    freq = Counter(corpus_tokens)
    freq = OrderedDict(freq.most_common())
    for key, value in freq.items():
        # if key not in self.stop_words:
        word2idx[key] = idx_counter
        idx2word[idx_counter] = key
        idx_counter += 1
    print("vocab size:", len(word2idx))
    return word2idx, idx2word, freq
```

3. From the token frequency distribution curve, we notice that the given corpus fairly follows the **Zipf's law** which states that the frequency of a given word is dependent on the inverse of its rank where rank is the index we're associating with each token in our vocabulary (highest frequency being rank 1). This means that the extreme ranks represent high frequency stopwords and least frequently observed words which we can get rid of from the vocabulary.



From the cumulative fraction covered curve, we notice that if we remove frequencies up to 55 we cover 91% of the tokens in our vocab whereas if we remove word frequencies up to 44, we cover 92% of the tokens in vocab. Putting cutoff point anywhere after this slowly changes the fraction covered and hence we choose to cutoff by removing frequencies 1 to 55 to cover about 91% tokens in the dataset whereas the rest of the 9% would correspond to 'UNK'.



2. Frequency-Based Word Vectors - PPMI

1. From the equation of PMI, PMI is directly proportional to the probability of seeing words w_i and w_j together whereas it is inversely proportional to probabilities of seeing words w_i, w_j independently. Thus, PMI tells us if the words w_i, w_j co-occur more than if they were independent.
   PMI ranges between:

$$-\infty \leq \text{ PMI } (w_i, w_j) \leq \text{ min } [-\log p(w_i), -\log p(w_j)]$$

   Positive PMI means,

$$\frac{p(w_i, w_j)}{p(w_i)p(w_j)} > 1$$

$$\implies p(w_i, w_j) > p(w_i)p(w_j)$$

$$\implies w_i \text{ and } w_j \text{ occur more together than individually (like bread and butter)}$$

   Zero PMI means,

$$p(w_i, w_j) = 0 \text{ showing that } w_i \text{ and } w_j \text{ are independent and never co-occur}$$

   Negative PMI means,

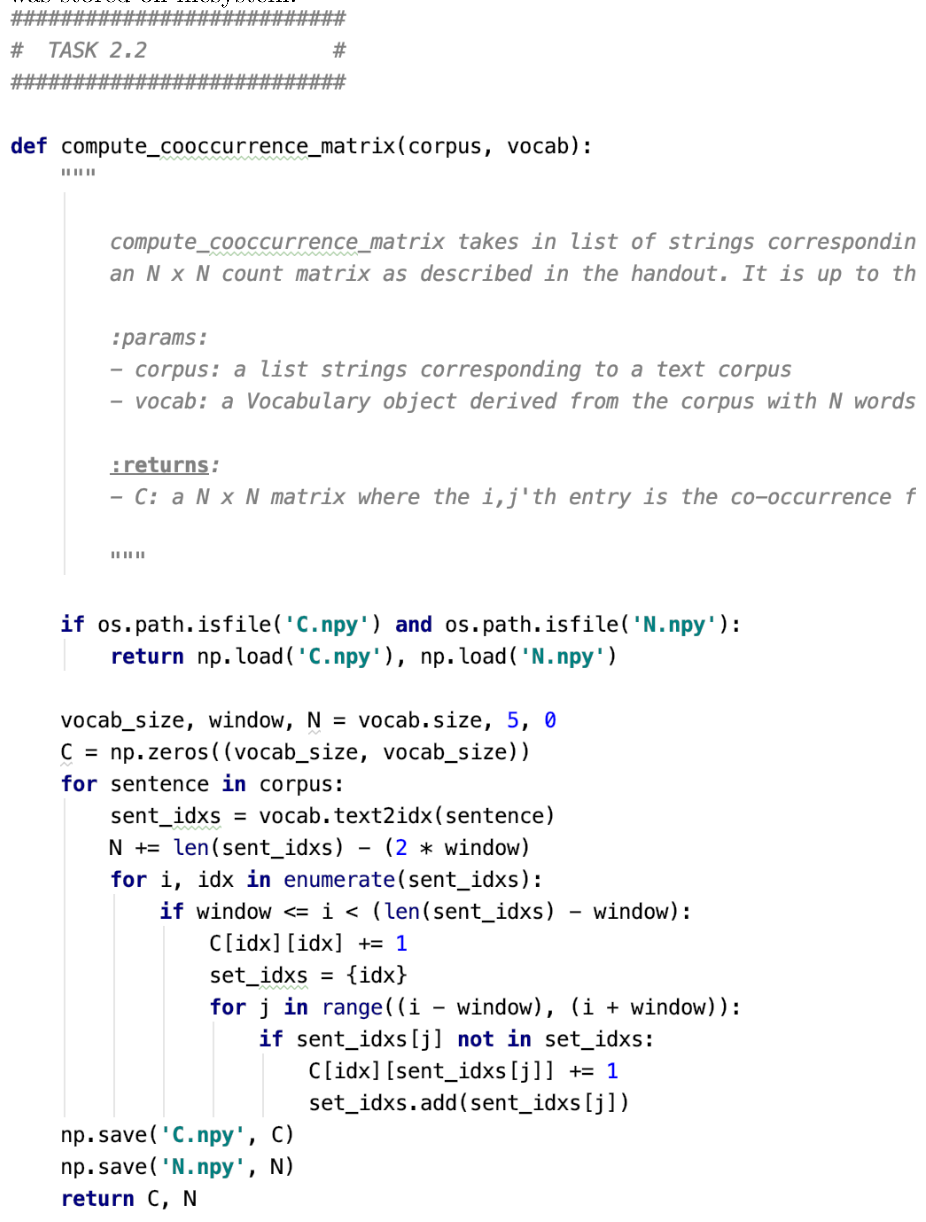
$$\frac{p(w_i, w_j)}{p(w_i)p(w_j)} < 1$$

$$\implies p(w_i, w_j) < p(w_i)p(w_j)$$

$$\implies w_i \text{ and } w_j \text{ or one of them tend to occur more individually}$$

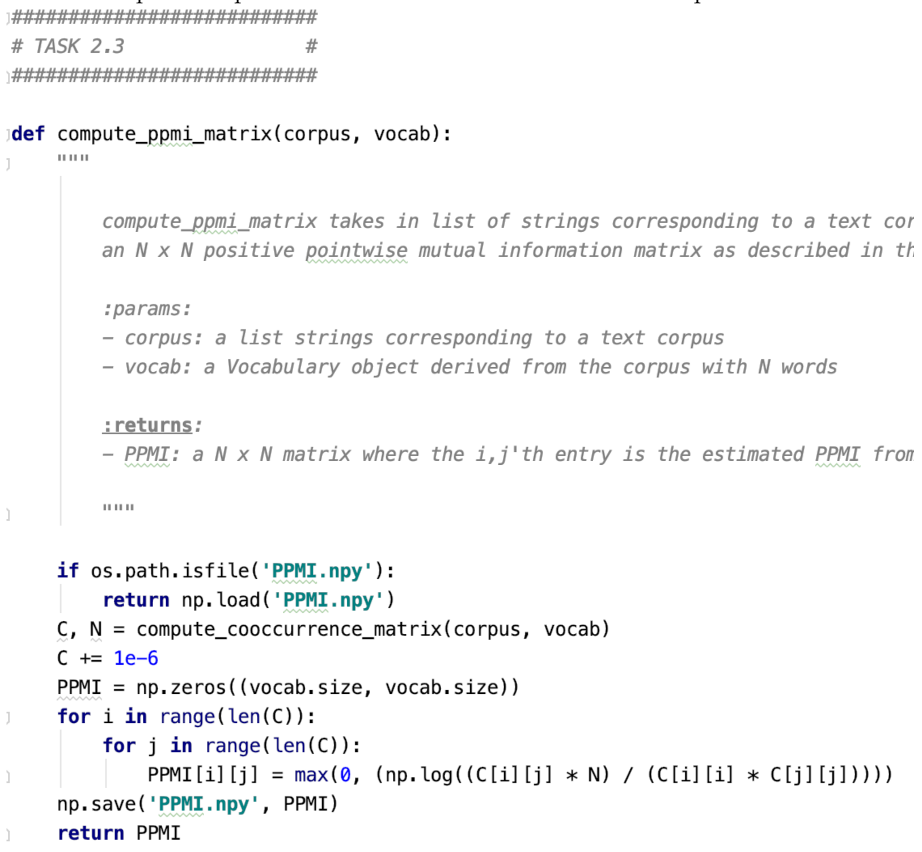
   Negative PMI would thus show uninformative co-occurrences like ”the bread” where ”the” occurs with many other words too. Also, for infrequent words, there is very less data to accurately determine negative PMI. This is probably why we drop the negative PMIs and use PPMIs which enable us in covering situations with informative co-occurrence

2. Cooccurrence matrix was computed using a sliding window of 5 on either sides of a word/token, making it an 11 word long window. Whilst choosing the window, sizes 2(2+1+2), 3(3+1+3), 4(4+1+4), 5(5+1+5), 6(6+1+6) were tried out. In order to avoid recomputations of cooccurrence matrix, the intermediate output was stored on filesystem.

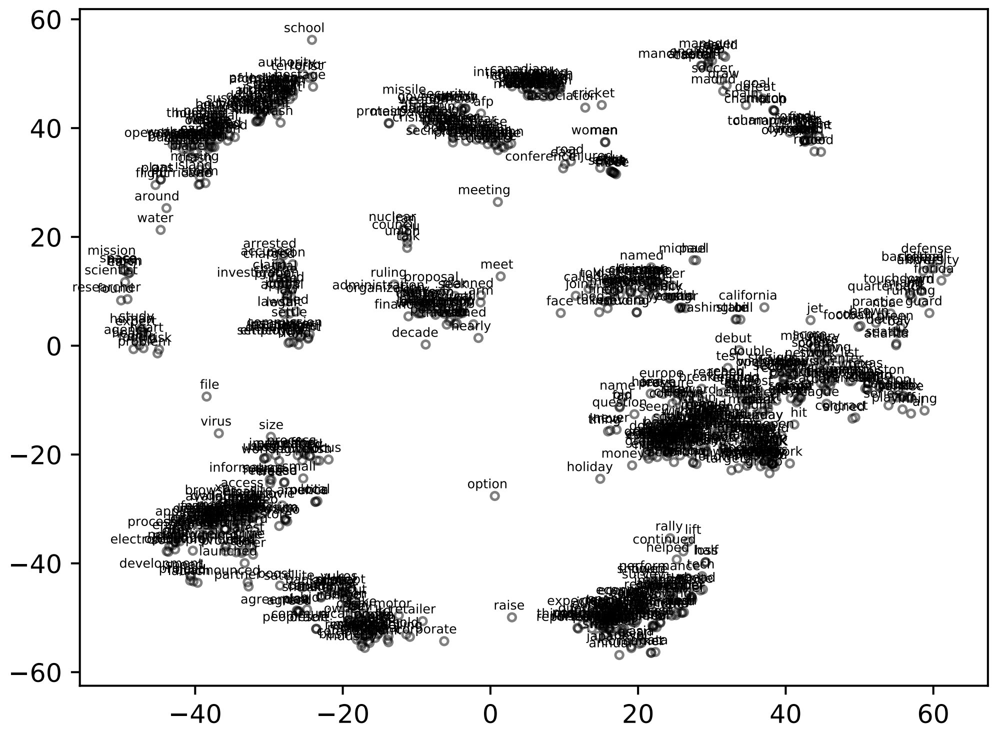


```
###########################
#  TASK 2.2               #
###########################

def compute_cooccurrence_matrix(corpus, vocab):
    """

        compute_cooccurrence_matrix takes in list of strings correspondin
        an N x N count matrix as described in the handout. It is up to th

        :params:
        - corpus: a list strings corresponding to a text corpus
        - vocab: a Vocabulary object derived from the corpus with N words

        :returns:
        - C: a N x N matrix where the i,j'th entry is the co-occurrence f

        """

    if os.path.isfile('C.npy') and os.path.isfile('N.npy'):
        return np.load('C.npy'), np.load('N.npy')

    vocab_size, window, N = vocab.size, 5, 0
    C = np.zeros((vocab_size, vocab_size))
    for sentence in corpus:
        sent_idxs = vocab.text2idx(sentence)
        N += len(sent_idxs) - (2 * window)
        for i, idx in enumerate(sent_idxs):
            if window <= i < (len(sent_idxs) - window):
                C[idx][idx] += 1
                set_idxs = {idx}
                for j in range((i - window), (i + window)):
                    if sent_idxs[j] not in set_idxs:
                        C[idx][sent_idxs[j]] += 1
                        set_idxs.add(sent_idxs[j])
    np.save('C.npy', C)
    np.save('N.npy', N)
    return C, N
```

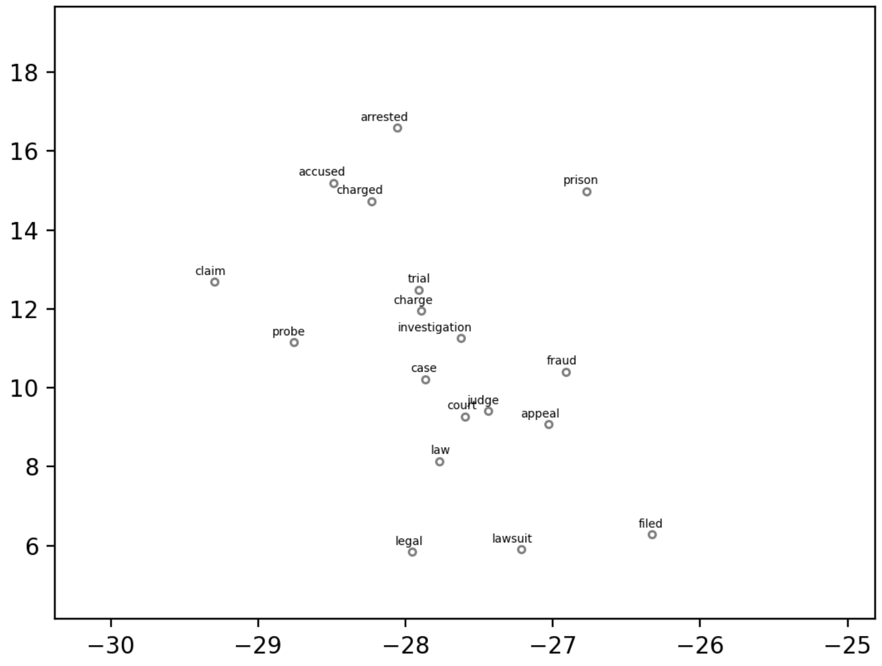
3. Matrix for positive point-wise mutual information was computed as follows:



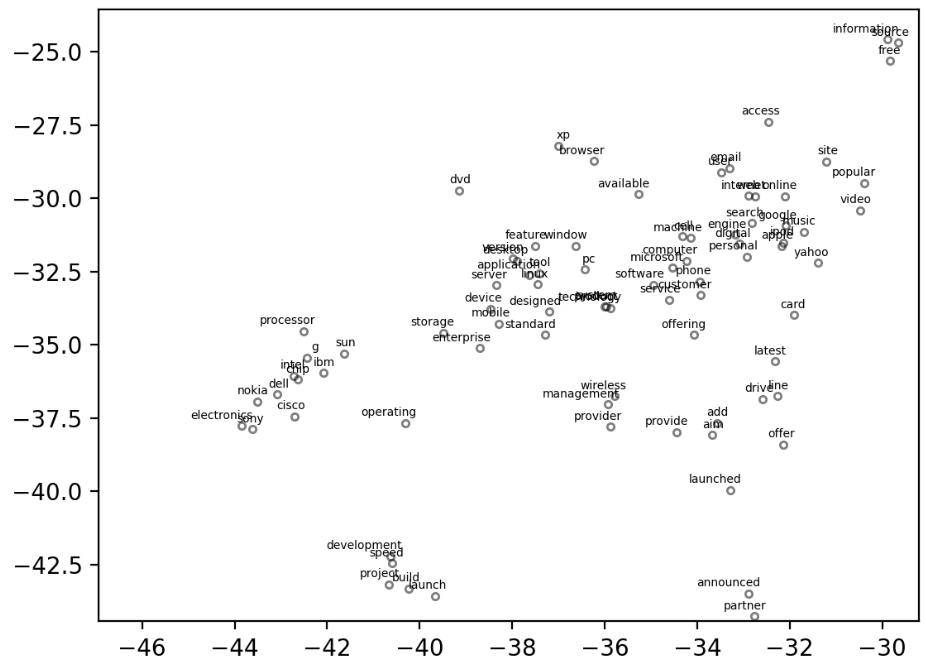
```
##########################
# TASK 2.3               #
##########################

def compute_ppmi_matrix(corpus, vocab):
    """

        compute_ppmi_matrix takes in list of strings corresponding to a text cor
        an N x N positive pointwise mutual information matrix as described in th

        :params:
        - corpus: a list strings corresponding to a text corpus
        - vocab: a Vocabulary object derived from the corpus with N words

        :returns:
        - PPMI: a N x N matrix where the i,j'th entry is the estimated PPMI from

        """

    if os.path.isfile('PPMI.npy'):
        return np.load('PPMI.npy')
    C, N = compute_cooccurrence_matrix(corpus, vocab)
    C += 1e-6
    PPMI = np.zeros((vocab.size, vocab.size))
    for i in range(len(C)):
        for j in range(len(C)):
            PPMI[i][j] = max(0, (np.log((C[i][j] * N) / (C[i][i] * C[j][j]))))
    np.save('PPMI.npy', PPMI)
    return PPMI
```



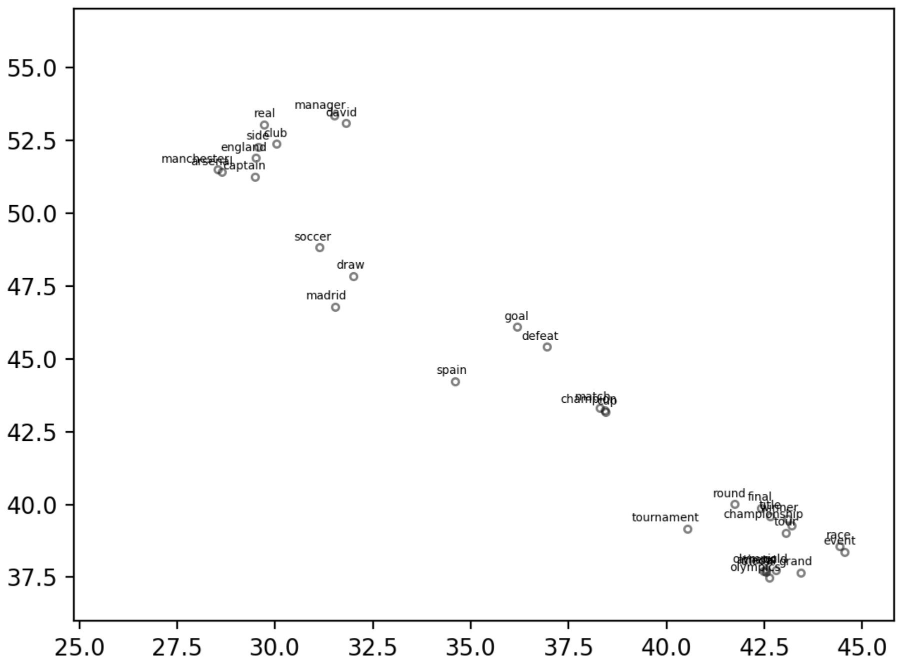
1. Law:



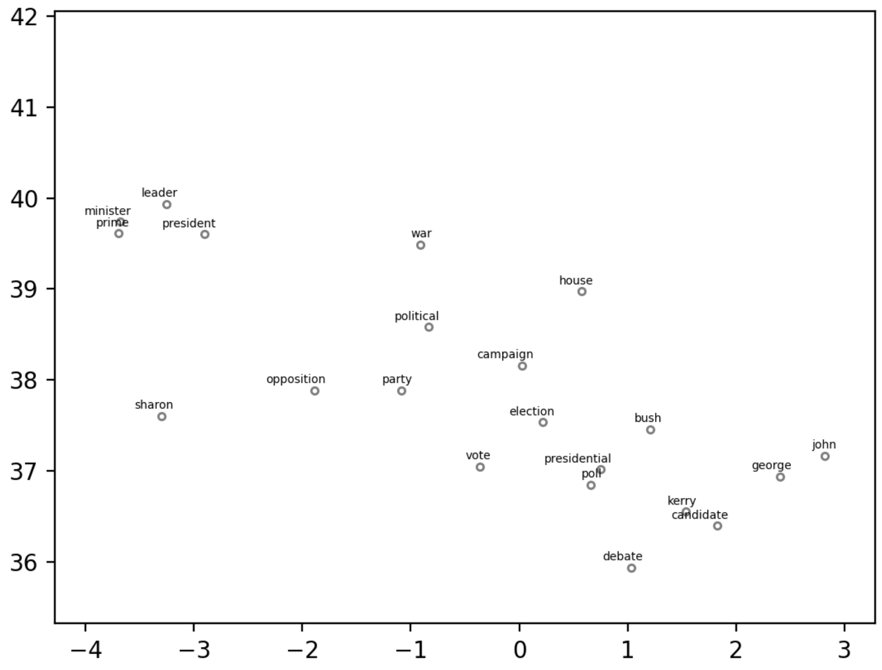
2. Technology:



3. Sports:



4. Political:



3. Learning-Based Word Vectors - GloVe

1.
$$J_B = \sum_{(i_m, j_m) \in B} f(C_{i_m j_m})(w_{i_m}^T \tilde{w}_{j_m} + b_{i_m} + \tilde{b}_{j_m} - \log C_{i_m j_m})^2 \tag{0.1}$$

The gradients of the objective J_B in (0.1) with respect to the model parameters $w_i, \tilde{w}_j, b_i$, and $\tilde{b}_j$ are as follows:

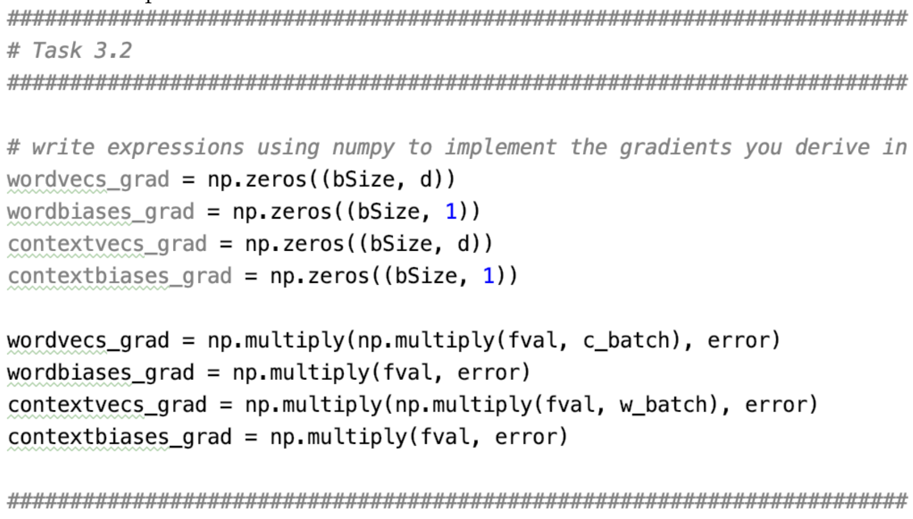
$$\Delta_{w_i} J_B = \sum_{j \in B} f(C_{ij}) \tilde{w}_j (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log C_{ij}) \tag{0.2}$$

$$\Delta_{\tilde{w}_j} J_B = \sum_{i \in B} f(C_{ij}) w_i (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log C_{ij}) \tag{0.3}$$

$$\Delta_{b_i} J_B = \sum_{j \in B} f(C_{ij}) (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log C_{ij}) \tag{0.4}$$

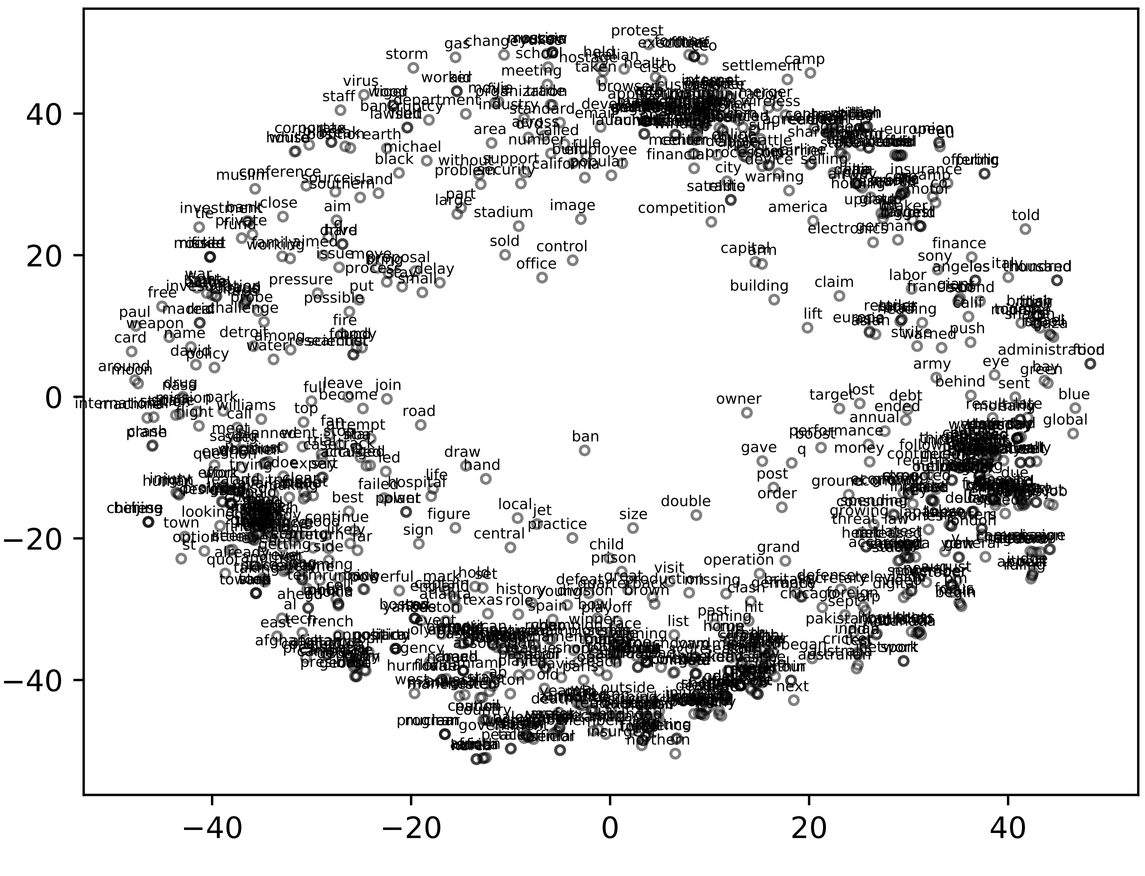
$$\Delta_{\tilde{b}_j} J_B = \sum_{i \in B} f(C_{ij}) (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log C_{ij}) \tag{0.5}$$

2. Gradient implementation:

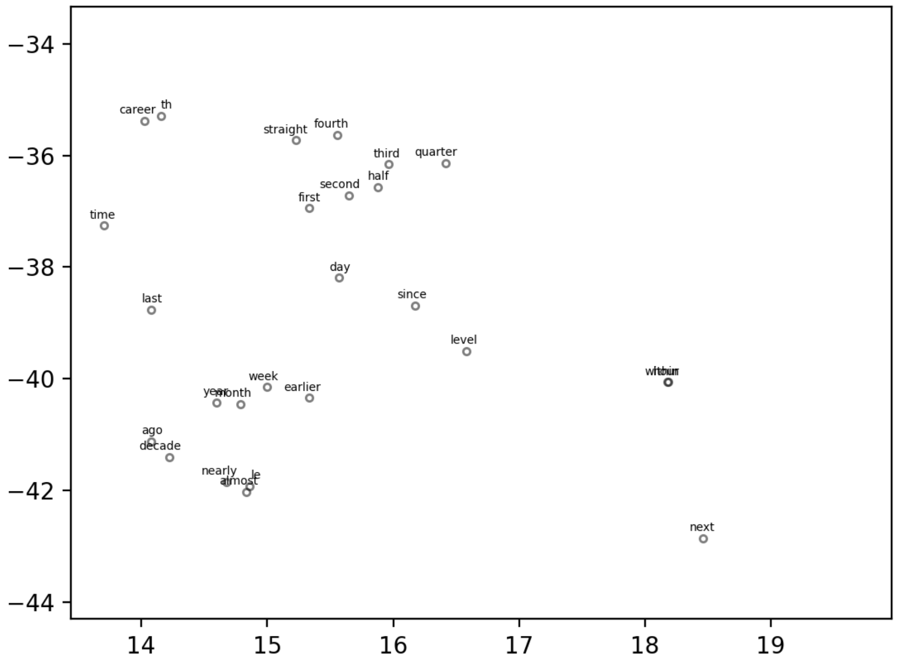


```
######################################################################
# Task 3.2
######################################################################

# write expressions using numpy to implement the gradients you derive in
wordvecs_grad = np.zeros((bSize, d))
wordbiases_grad = np.zeros((bSize, 1))
contextvecs_grad = np.zeros((bSize, d))
contextbiases_grad = np.zeros((bSize, 1))

wordvecs_grad = np.multiply(np.multiply(fval, c_batch), error)
wordbiases_grad = np.multiply(fval, error)
contextvecs_grad = np.multiply(np.multiply(fval, w_batch), error)
contextbiases_grad = np.multiply(fval, error)

######################################################################
```

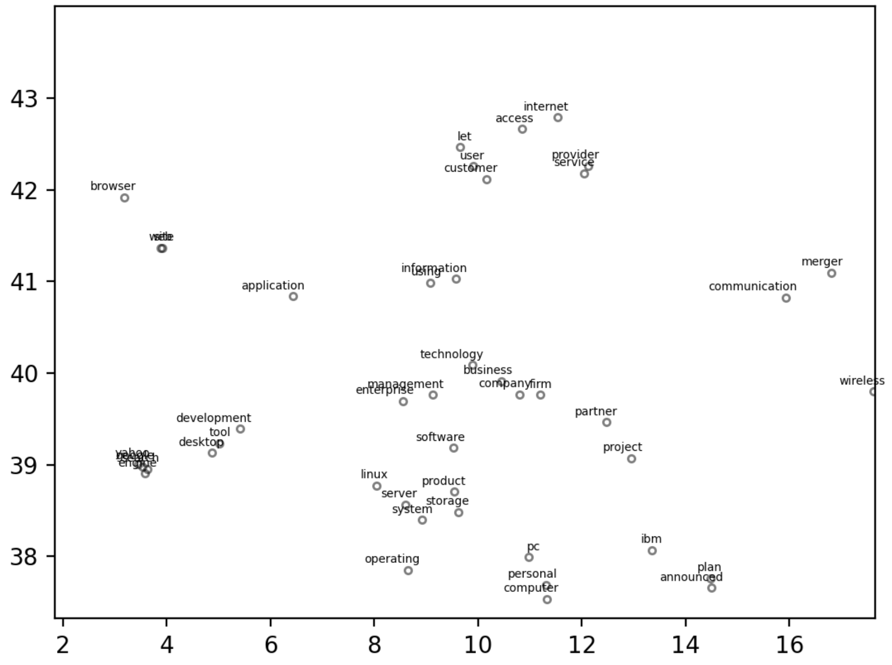
3. During training using GloVe vectors with learning rate 0.1, 6372 words in the vocab, and 3275651 non-zero entries in the count matrix, the loss went from 0.95 to 0.09 very quickly within the first epoch after which the loss was observed to be stably going down to 0.04 up to the 20^{th} epoch. It converged to about 0.036 after 30 epochs but the TSNE plot after 20 epochs seemed to be as semantically meaningful as that of the one after 30 epochs. For minimizing loss, TSNE after 30 epochs was generated. This ring-shaped GloVe plot seemed to have similar clusters of words compared to the previous approach however difference was that these clusters appeared a bit overlapping due to having the ring structure. Following is the the ring-shaped TSNE GloVe plot after 30 epochs and loss around 0.037:



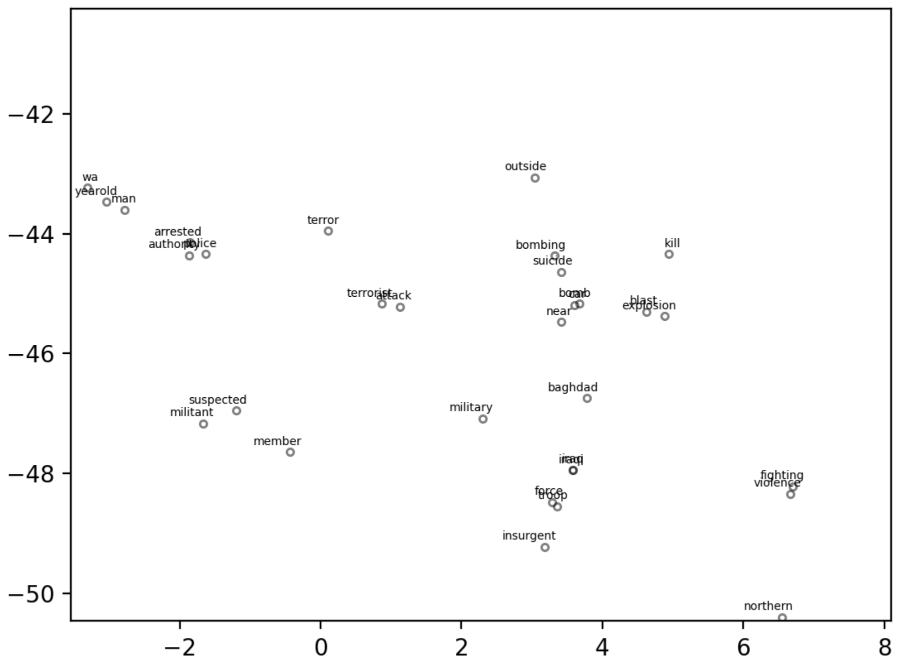
1. Time:



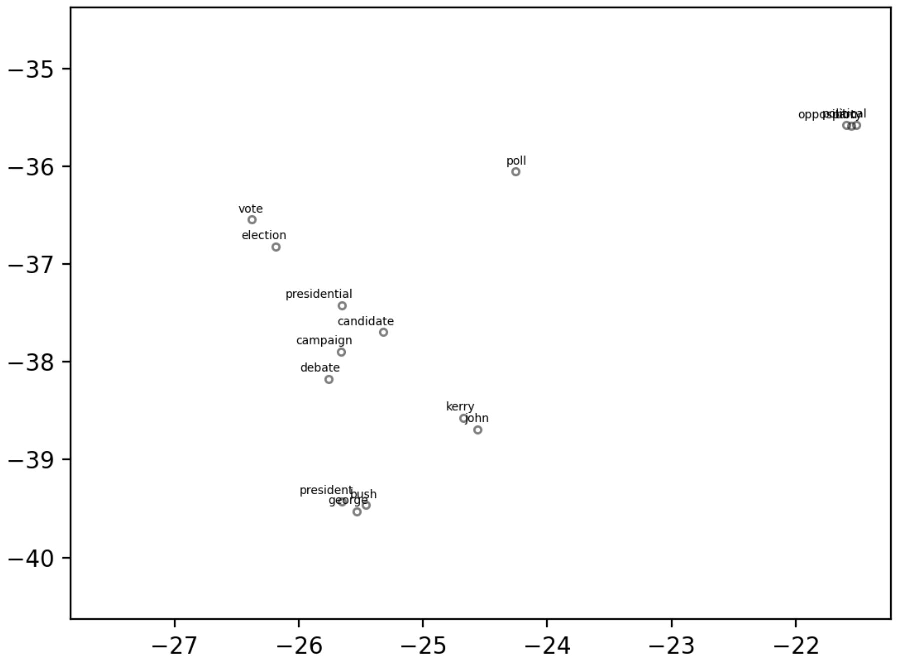
2. Technology:



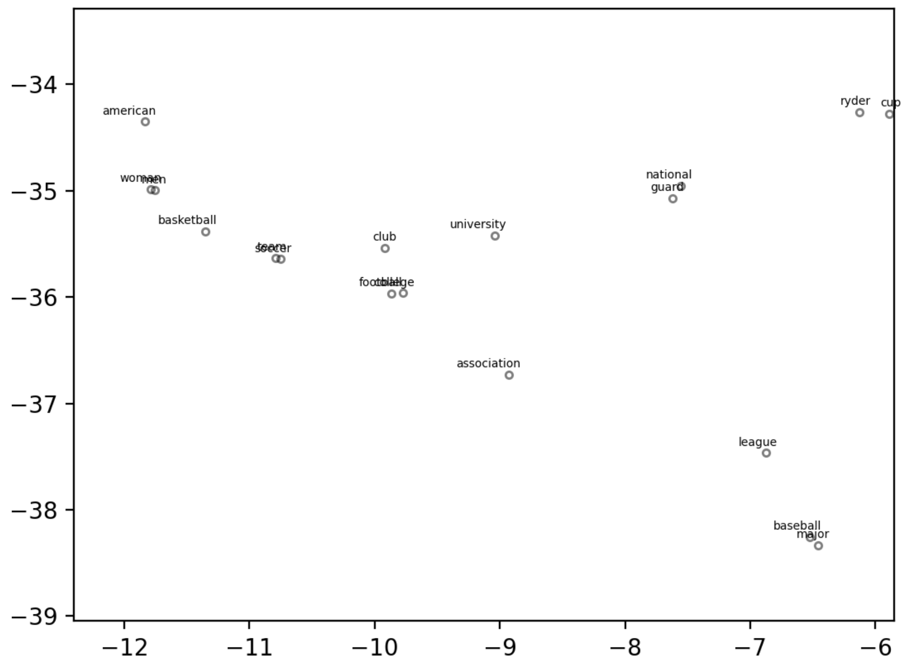
3. Military:



4. Political:

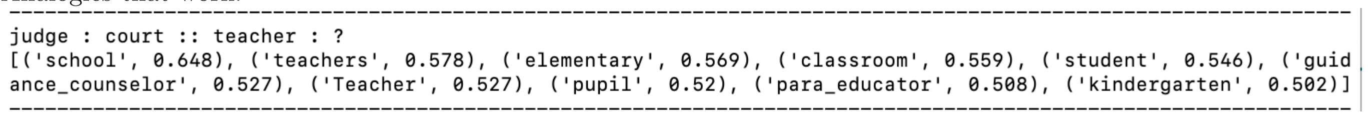


5. Sports:


american
woman
men
basketball
team
soccer
club
football
college
university
association
national
guard
ryder
cup
league
baseball
major
−34
−35
−36
−37
−38
−39
−12
−11
−10
−9
−8
−7
−6

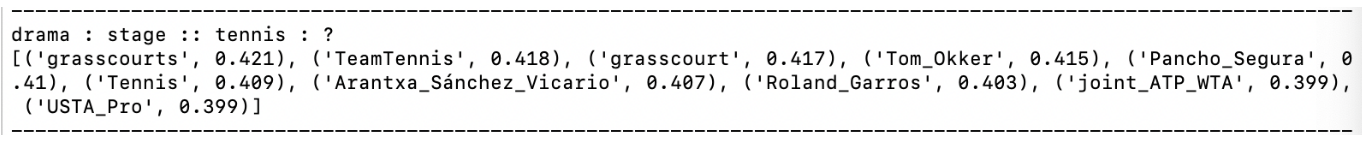
4. Exploring Learned Biases in word2vec Vectors

1. Analogies that work:



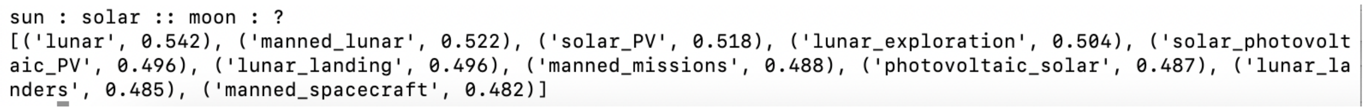
```
----------------------------------------------------------------------------------------------------------
judge : court :: teacher : ?
[('school', 0.648), ('teachers', 0.578), ('elementary', 0.569), ('classroom', 0.559), ('student', 0.546), ('guid
ance_counselor', 0.527), ('Teacher', 0.527), ('pupil', 0.52), ('para_educator', 0.508), ('kindergarten', 0.502)]
----------------------------------------------------------------------------------------------------------
```

   Judge works at the court whereas teacher teaches at school. Occupation to workplace relationship.



```
----------------------------------------------------------------------------------------------------------
drama : stage :: tennis : ?
[('grasscourts', 0.421), ('TeamTennis', 0.418), ('grasscourt', 0.417), ('Tom_Okker', 0.415), ('Pancho_Segura', 0
.41), ('Tennis', 0.409), ('Arantxa_Sánchez_Vicario', 0.407), ('Roland_Garros', 0.403), ('joint_ATP_WTA', 0.399),
 ('USTA_Pro', 0.399)]
----------------------------------------------------------------------------------------------------------
```

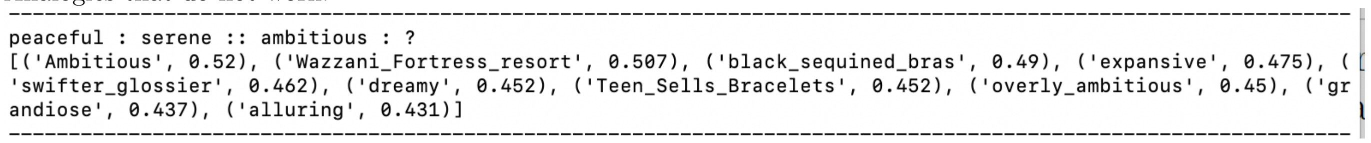
   Drama is performed on stage whereas tennis is played on courts.



```
sun : solar :: moon : ?
[('lunar', 0.542), ('manned_lunar', 0.522), ('solar_PV', 0.518), ('lunar_exploration', 0.504), ('solar_photovolt
aic_PV', 0.496), ('lunar_landing', 0.496), ('manned_missions', 0.488), ('photovoltaic_solar', 0.487), ('lunar_la
nders', 0.485), ('manned_spacecraft', 0.482)]
```

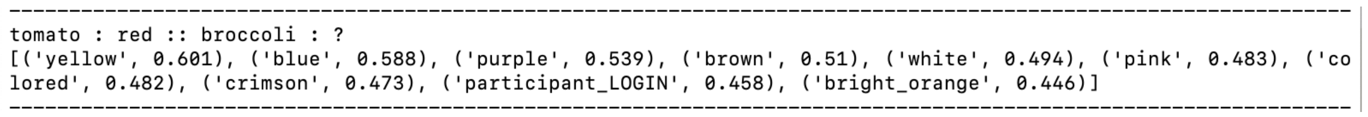
   Noun to adjective form.

2. Analogies that do not work:



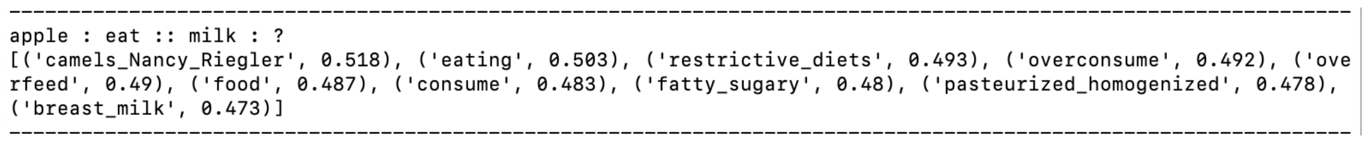
```
----------------------------------------------------------------------------------------------------------
peaceful : serene :: ambitious : ?
[('Ambitious', 0.52), ('Wazzani_Fortress_resort', 0.507), ('black_sequined_bras', 0.49), ('expansive', 0.475), (
'swifter_glossier', 0.462), ('dreamy', 0.452), ('Teen_Sells_Bracelets', 0.452), ('overly_ambitious', 0.45), ('gr
andiose', 0.437), ('alluring', 0.431)]
----------------------------------------------------------------------------------------------------------
```

   Synonym to ambitious wasn't found.



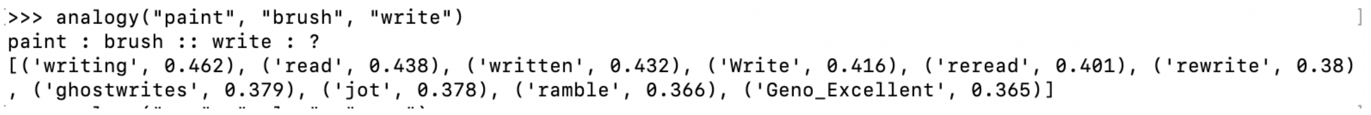
```
----------------------------------------------------------------------------------------------------------
tomato : red :: broccoli : ?
[('yellow', 0.601), ('blue', 0.588), ('purple', 0.539), ('brown', 0.51), ('white', 0.494), ('pink', 0.483), ('co
lored', 0.482), ('crimson', 0.473), ('participant_LOGIN', 0.458), ('bright_orange', 0.446)]
----------------------------------------------------------------------------------------------------------
```

   Broccoli is green in color which is nowhere in the results.



```
----------------------------------------------------------------------------------------------------------
apple : eat :: milk : ?
[('camels_Nancy_Riegler', 0.518), ('eating', 0.503), ('restrictive_diets', 0.493), ('overconsume', 0.492), ('ove
rfeed', 0.49), ('food', 0.487), ('consume', 0.483), ('fatty_sugary', 0.48), ('pasteurized_homogenized', 0.478),
('breast_milk', 0.473)]
----------------------------------------------------------------------------------------------------------
```

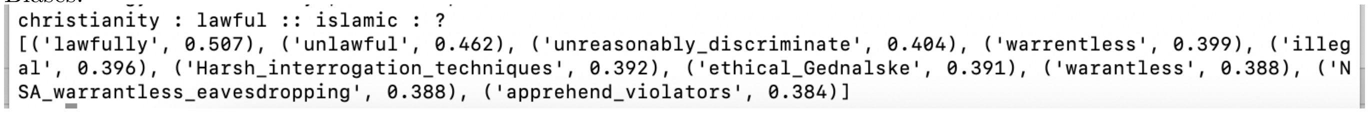
   Apple is to eat and milk is to drink, which is not present, although consume is close enough.



```
>>> analogy("paint", "brush", "write")
paint : brush :: write : ?
[('writing', 0.462), ('read', 0.438), ('written', 0.432), ('Write', 0.416), ('reread', 0.401), ('rewrite', 0.38)
, ('ghostwrites', 0.379), ('jot', 0.378), ('ramble', 0.366), ('Geno_Excellent', 0.365)]
```

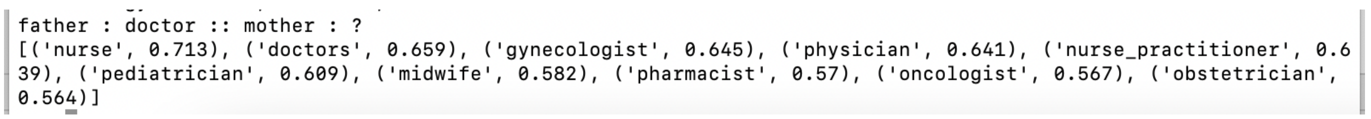
   Painting is done with a brush, writing with a pen.

3. Biases:



```
christianity : lawful :: islamic : ?
[('lawfully', 0.507), ('unlawful', 0.462), ('unreasonably_discriminate', 0.404), ('warrentless', 0.399), ('illeg
al', 0.396), ('Harsh_interrogation_techniques', 0.392), ('ethical_Gednalske', 0.391), ('warantless', 0.388), ('N
SA_warrantless_eavesdropping', 0.388), ('apprehend_violators', 0.384)]
```

   There's a terrible religious bias here which associates lawful with christianity and unlawful, illegal with muslim.



```
father : doctor :: mother : ?
[('nurse', 0.713), ('doctors', 0.659), ('gynecologist', 0.645), ('physician', 0.641), ('nurse_practitioner', 0.6
39), ('pediatrician', 0.609), ('midwife', 0.582), ('pharmacist', 0.57), ('oncologist', 0.567), ('obstetrician',
0.564)]
```

   Father is to doctor then mother is to nurse, nurse_parctitioner, midwife. Again, females are associated with homemaker a lot.

4. There have been biases, prejudices, discriminations, stereotypes in the society since aeons. Although the current century is much more evolved than what it was before, the input data to our machine learning models carries such small to large biases. Machine learning models tend to amplify such biases in the data which is perturbing. The models weren't even fed the information that queen is feminine and king is masculine but, the relationships between words were be captured by vector algebra. Thus the bias happens because any bias in the articles that make up the word2vec corpus is inevitably captured in the geometry of the vector space. All the job portals as of today ask to fill details about religion, race, gender, veteran, disability status, etc. If word2vec were used in deriving insights from a pool of applicants, there could be very wrong inferences and amplified biases in the learned model. If word2vec were used in recommendation systems, humans will have direct impact on their minds.