Homework 1: Vector space word similarity (CS 2731 Fall 2023)

Due 2023-09-17, 11:59pm (extended from 2023-09-14 initial deadline). *Instructions last updated 2023-09-13.*

In this assignment, you'll build representations for documents and words based on the bag-of-words model. You'll implement 2 popular weighting schemes for these vectors: tf-idf and PPMI, both discussed in Chapter 6 of the textbook. Then you'll compare these weighting schemes on learning word similarity and apply one of them, PPMI, to examine social bias in an NLP corpus.

Datasets and skeleton code

Here are the materials that you should download for this assignment:

- Skeleton Python code. You will need to have a Python environment with scipy and numpy packages. Some functions are stubs in the python code. You will need to fill them out.
- CSV of the complete works of Shakespeare
- Vocab of the complete works of Shakespeare
- List of all plays in the dataset
- SNLI corpus
  - This corpus is selections from SNLI, a corpus used for the NLP task of "natural language inference" (see Bowman et al. 2015 dataset paper). Each line contains a sentence that is either a "premise" (an image caption) or a "hypothesis" produced by annotators to be in a certain logical relation with the associated premise (entailment, neutral, contradiction). You don't need to worry about these details, but the `sentenceID` column is a unique index for each sentence and `captionID` is an ID for all sentences associated with same caption/premise.
- List of identity labels from Rudinger et al. 2017

Part 1: Vector spaces

1.1 Term-document matrix

Write code to compile a term-document matrix for Shakespeare's plays, following the description in the textbook:

> *In a term-document matrix, each row represents a word in the vocabulary and each column represents a document from some collection. The figure below shows a small selection from a term-document matrix showing the occurrence of four words in four plays by Shakespeare. Each cell in this matrix represents the number of times a particular word (defined by the row) occurs in a particular document (defined by the column). Thus* ***clown*** *appeared 117 times in* ***Twelfth Night***

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	5	117	0	0

The dimensions of your term-document matrix will be the number of documents D (in this case, the number of Shakespeare's plays that we give you in the corpus) by the number of unique word types $|V|$ in that collection. The columns represent the documents, and the rows represent the words, and each cell represents the frequency of that word in that document.

Tasks for section 1.1

- Fill out the function stub `create_term_document_matrix`.

1.2 Term-Context Matrix

Instead of using a term-document matrix, a more common way of computing word similarity is by constructing a term-context matrix (also called a term-term or word-word matrix), where columns are labeled by words rather than documents. The dimensionality of this kind of a matrix is $|V|$ by $|V|$. Each cell represents how often the word in the row (the target word) co-occurs with the word in the column (the context) in a training corpus. You can decide when it makes sense for a word to co-occur with itself in the term-context matrix. That is, will the cell for when the same word is target and context always stay 0?

Tasks for section 1.2

- Implement the `create_term_context_matrix` function. This function specifies the size word window around the target word that you will use to gather its contexts. For instance, if you set that variable to be 4, then you will use 4 words to the left of the target word, and 4 words to its right for the context. In this case, the cell represents the number of times in Shakespeare's plays the column word occurs in +/-4 word window around the row word.

1.3 Evaluating vector spaces

So far we have created 2 vector spaces for the words in Shakespeare, one with a dimension of D and another of dimension $|V|$. Now we will try to evaluate how good our vector spaces are. We can do this with an intrinsic evaluation approach by seeing what words within the vocab are most similar to each other/are synonyms with each other and assessing if the output is reasonable. Implement the `rank_words` function which will take a target word index and return a list sorted from most similar to least similar using the cosine similarity metric. For the purposes of the assignment, let's just look at the top 10 words that are most similar to a target word between both the term-document matrix and the term-context matrix (with a window size of your choice). Are those 10 words good synonyms? The skeleton code provides an example of using `rank_words` and looking at similar words using the word 'juliet'. One example won't be enough so pick out at least 4 more words from the vocab as you answer these questions.

Tasks for section 1.3

- Implement `rank_words`

For the report:

- In our term-document matrix, the rows are word vectors of D dimensions. Do you think that's enough to represent the meaning of words?
- Which vector space (term-document or term-context) produce similar words that make more sense than others and why do you think that is the case?
- Consider any decisions you made in the prior sections when implementing your functions, such as whether you allowed a target word to co-occur with itself as a context word, and which window size you chose for the term-context matrix. How might any decisions you make impact our results now?

1.4 Weighting terms with tf-idf and PPMI

Your term-context matrix contains the raw frequency of the co-occurrence of two words in each cell and your term-document matrix contains the raw frequency of words in each of the documents. Raw frequency turns out not to be the best way of measuring the association between words. There are several methods for weighting words so that we get better results.

Tasks for section 1.4

- Take your term-document matrix and implement the weighting schemes *Term frequency inverse document frequency (tf-idf)* and *Positive pointwise mutual information* which are defined in Sections 6.5-6.6 of the textbook. These are the function stubs `create_tf_idf_matrix` and `create_ppmi_matrix`.

For the report:

- Redo the analysis in section 1.3 with ranked words and compare using tf-idf and PPMI with unweighted term-document and term-context matrices.
- Discuss findings from comparing approaches. Do some approaches appear to work better than others, i.e produce better synonyms? Do any interesting patterns emerge?
- How does weighting with tf-idf compare to using the unweighted term-document matrix? How does weighting with PPMI compare with using the unweighted term-context matrix? How does term-context/PPMI compare to term-document/TF-IDF? Include results and discussion.

Part 2

In this part, you will measure associations between words in a commonly used NLP corpus, SNLI, and comment on the potential for encoding problematic social biases. There is no skeleton code for this section, but you can reuse code from Part 1.

Tasks for Part 2

- First, write a loader for text from the SNLI corpus into a similar format as the Shakespeare corpus was loaded. You can use the `sentenceID` column as the document name, though this will not be as important since you'll only be building a term-context matrix. You'll still want to tokenize and lowercase the input as was done with the Shakespeare corpus.
- Build a term-context matrix in a similar fashion as with the Shakespeare corpus and apply PPMI weighting. You can choose the size of the context window. You can also use the whole sentence as the context. If this matrix is too big or taking too long to calculate, filter to just words that occur over some frequency threshold in the entire corpus.

For the report:

- With that PPMI-weighted term-context matrix, find the vectors for identity labels in the provided list. Look at the top associated words (by PPMI) for at least 4 identity labels of your choice. Do you see any that may reflect social stereotypes? It is helpful to compare the top PMI words for certain identity terms with other related ones (such as men compared with women). Discuss and provide selected results in the report.
- Qualitative analysis: For at least 4 different identity labels, dig into the contexts that leads to high PMI association with other words, especially for any words that show social bias if you found that. 1st-order similarity: find specific examples from the dataset where an identity label occurs with a top-associated term that shows some social bias or does not. This might not occur; if not, you can look at 2nd-order similarity in which the two words occur with similar context words. Sample the contexts/documents in which the 2 words occur separately. Do they occur with the same set of context words? This can also be examined by looking at the vectors for the identity term and the highly associated other term in the term-context matrix. Do these share high values in certain dimensions that correspond to certain context words? Provide selected results and discuss findings in the report. Do you see evidence for representational harms (see below) learned by a bag-of-words model of this SNLI corpus? If so, which type do you see? Provide examples that support your conclusions. If you don't find any potential harms, provide examples of what you examined and how you interpreted those associations.
- Here is a description of *representational harms* in machine learning from Blodgett et al. 2020.

> *Representational harms arise when a system (e.g., a search engine) represents some social groups in a less favorable light than others, demeans them, or fails to recognize their existence altogether.*

Types of representational harms from Blodgett et al. 2020 include:

> *1. Stereotyping that propagates negative generalizations about particular social groups*
> *2. Differences in system performance for different social groups, language that misrepresents the distribution of different social groups in the population, or language that is denigrating to particular social groups.*

Deliverables

- Your implementations for the functions in the skeleton code `hw1_skeleton_{your pitt id}.py`. You are welcome to put code for Part 2 in the same or a different file. If it's different, please where it is in the README.txt.
  - You are welcome to import any packages you need but please don't modify the function that has already been implemented.
- Your report with results and answers to questions in Part 1 and Part 2, named `report_{your pitt id}.pdf`.
- A README.txt file explaining
  - how to run your code
  - the computing environment you used; what programming language you used and the major and minor version of that language; what packages did you use in case we replicate your experiments (a `requirements.txt` file for setting up the environment may be useful if there are many packages).
  - any additional resources, references, or web pages you've consulted
  - any person with whom you've discussed the assignment and describe the nature of your discussions
  - any generative AI tool used, and how it was used
  - any unresolved issues or problems

Please submit all of this material on Canvas. We will grade your report and attempt to run your code.

Acknowledgments

This assignment is adapted from Prof. Diane Litman and Prof. Mark Yatskar, as well from Rudinger et al. 2017.