Ethics for NLP: Spring 2022
Homework 1

TECHNISCHE UNIVERSITÄT DARMSTADT

Due until Thursday, 19.05.2022 at 11:30am

Submission Guidelines for Homework

- This homework worth 20 Points.
- You may use the `.ipynb` file as a template.
- Submit your code and answers in a single `.ipynb` notebook. (Submit through Moodle.)
- Extra credit shall be given to well-structured submissions.
- In case of questions or remarks, please contact:
  - Aniket Pramanick, pramanick@ukp.informatik.tu-darmstadt.de

1 **Analysis of Bias in Word Vectors** (20 Points)

Before you start make sure you read the Submission Guideline instructions associated to this homework for important setup and submission information. Additionally, we encourage you to use the notebook provided with this homework as a template, as we have already put a lot of code in it. Also, this will give you a head start on the assignment.

1.1 **Goals**

It is important to be cognizant and explore the implications of underlying biases in the crowd sourced data used for various Natural Language Processing(NLP) systems. Bias can be dangerous because it can reinforce stereotypes through applications that employ these systems.

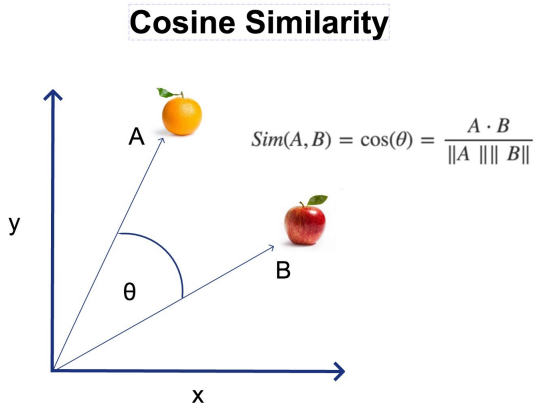
The goal of this assignment is to understand how biases are reflected implicitly in the word embeddings.

1.2 **Word Vectors**

Word Vectors (often interchangeably used as *Word Embeddings*) are often used as fundamental component for downstream NLP tasks, e.g. question answering, text generation, translation, etc., so it is important to gain some insights into their strengths and weaknesses.

Here, you will explore two types of word vectors: those derived from co-occurrence matrices, and the prediction-based word vectors.

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.
Think n-dimensional vectors as points in n-dimensional space. If we take this perspective the angle between the two vectors helps to quantify the amount of space "we must travel" to get between these two points. From trigonometry we know that:



Instead of computing the actual angle, we can leave the similarity in terms of $similarity = cos\theta$. Formally, cosine similarity between two vectors A and B is defined as

$$sim(A,B) = \frac{A \cdot B}{\|A\|\|B\|}$$

1.2.1 Count-Based Word Vectors

Most word vector models start from the following idea:

> " You shall know a word by the company it keeps "

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 or here).

1.2.1.1 Task I: Compute Co-Occurence Matrix (3 Points)

A co-occurence 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 includes the n preceding and n subsequent words in that document, i.e. words $w_{i-n}...w_{i-1}$ and $w_{i+1}...w_{i+n}$. 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 among all documents.

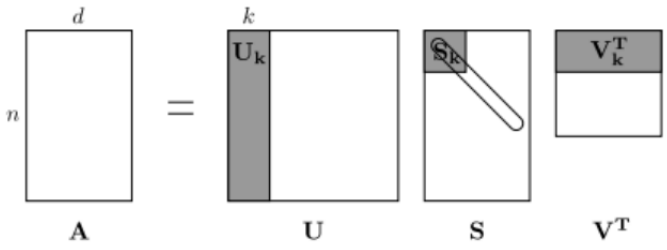
Example: Co-occurrence with Fixed Window n=1:

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
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 this 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.



Here, we will be using the Reuters (business and financial news) corpus. 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 here. We provide a read_corpus function that pulls out only articles from the "grain" (i.e. news articles about corn, wheat, 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.

For this task 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 and matrices.

1.2.1.2 Task II: Reduce Embedding Dimension (3 Points)

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.

1.2.1.3 Task III: Co-Occurrence Embeddings Plot and Analysis (Point 3)

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. In the future, a good way to make a plot is to look at the Matplotlib gallery, find a plot that looks somewhat like what you want, and adapt the code they give.
Now we will put together all the parts you have written! We will compute the co-occurrence matrix with fixed window of 4 (the default window size), over the Reuters "grain" corpus. Then we will use TruncatedSVD to compute 2-dimensional embeddings of each word. Additionally, we need to 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.

Plot the 2D GloVe embeddings for:

{'tonnes', 'grain', 'wheat', 'agriculture', 'corn', 'maize', 'export', 'department', 'barley', 'grains', 'soybeans', 'sorghum'}

- What clusters together in 2-dimensional embedding space?
- What doesn't cluster together that you might think should have?

1.2.2 Prediction-Based Word Vectors

More recently prediction-based word vectors have demonstrated better performance, such as word2vec and GloVe (which also utilizes the benefit of counts). Here, we shall explore the embeddings produced by GloVe. If you're feeling adventurous, challenge yourself and try reading GloVe's original paper.

1.2.2.1 Task IV: Reduce Embedding Dimension (3 Points)

Let's directly compare the GloVe embeddings to those of the co-occurrence matrix. In order to avoid running out of memory, we will work with a sample of 10000 GloVe vectors instead.

- Put 10000 Glove vectors into a matrix M.
- Reduce the vectors from 200-dimensional to 2-dimensional.

1.2.2.2 Task V: Embedding Plot Analysis (3 Points)

Plot the 2D GloVe embeddings for:

{'tonnes', 'grain', 'wheat', 'agriculture', 'corn', 'maize', 'export', 'department', 'barley', 'grains', 'soybeans', 'sorghum'}

- What clusters together in 2-dimensional embedding space?

- What doesn't cluster together that you think should have? How is the plot different from the one generated earlier from the co-occurrence matrix?
- What is a possible cause for the difference?

1.2.3 **Guided Analysis of Bias in Word Vectors (2.5 Points)**

It's important to be cognizant of the biases (gender, race, sexual orientation etc.) implicit in our word embeddings. Bias can be dangerous because it can reinforce stereotypes through applications that employ these models.
Examine

- which terms are most similar to "girl" and "toy" and most dissimilar to "boy", and
- which terms are most similar to "boy" and "toy" and most dissimilar to "girl". Point out the difference between the list of female-associated words and the list of male-associated words, and explain how it is reflecting gender bias.

1.2.4 **Independent Analysis of Bias in Word Vectors (2.5 Points)**

Find another case where some bias is exhibited by the vectors. Please briefly explain the example of bias that you discover.
Also, give one explanation of how bias gets into the word vectors. What is an experiment that you could do to test for or to measure this source of bias?