**Assignment 2: Vector Space Models (due 1 March 2019, 11.59 pm EST)**

The goal of this assignment is to implement many of the things you learned in the lecture on Vector Space Models.

The goals of this assignment are:

1. To create a term-document and term-context matrices from a given corpus
2. To implement weighting methods and similarity metrics
3. To use the resulting vectors to measure how similar some documents are to each other and how similar some words are to each other

The materials provided in this zip file are:

1. The skeleton code that contains the function that you should implement
2. Dataset containing (a) the complete works of Shakespeare (corpus separated by semicolons), (b) the vocabulary of Shakespeare, (c) the list of all plays in the dataset

**Term-Document Matrix**

You will implement a term-document matrix for Shakespeare’s plays. In a *term-document matrix*, each row represents a word in the vocabulary and each column represents a document. For example, this is 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** | **Julias Caesar** | **Henry V** |
| --- | --- | --- | --- | --- |
| **battle** | 1 | 1 | 8 | 15 |
| **soldier** | 2 | 2 | 12 | 36 |
| **fool** | 37 | 58 | 1 | 5 |
| **crown** | 5 | 117 | 0 | 0 |

The dimensions of your term-document matrix will be the number of documents (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.

In the skeleton code write a function to create\_term\_document\_matrix.

**Comparing plays**

Using the term-document matrix, you will figure out which plays are most similar to each other, by comparing the column vectors. We could even look for outliers to see if some plays are so dissimilar from the rest of the plays.

Each column is a |V| -dimensional vector. One of the most common similarity metric is the cosine of the angle between the vectors. The cosine, like most measures for vector similarity used in NLP, is based on the dot product operator from linear algebra, also called the inner product.

The dot product acts as a similarity metric because it will tend to be high just when the two vectors have large values in the same dimensions. Alternatively, vectors that have zeros in different dimensions (orthogonal vectors) will have a dot product of 0, representing their strong dissimilarity.

This raw dot-product, however, has a problem as a similarity metric: it favors long vectors. The dot product is higher if a vector is longer, with higher values in each dimension. More frequent words have longer vectors, since they tend to co-occur with more words and have higher co-occurrence values with each of them. The raw dot product thus will be higher for frequent words. But this is a problem; we would like a similarity metric that tells us how similar two words are regardless of their frequency.

The simplest way to modify the dot product to normalize for the vector length is to divide the dot product by the lengths of each of the two vectors. This normalized dot product turns out to be the same as the cosine of the angle between the two vectors.

The cosine value ranges from 1 for vectors pointing in the same direction, through 0 for vectors that are orthogonal, to -1 for vectors pointing in opposite directions. Since our term-document matrix contains raw frequency counts, it is non-negative, so the cosine for its vectors will range from 0 to 1. 1 means that the vectors are identical, 0 means that they are totally dissimilar.

Implement compute\_cosine\_similarity, and for each play in the corpus, score how similar each other play is to it. Then, for each play, print out the play that is closest to it in the vector space (ignoring self-similarity).

**How do I know if my rankings are good?**

Take a look at [this grouping of Shakespeare’s plays into Tragedies, Comedies and Histories](https://en.wikipedia.org/wiki/Shakespeare%27s_plays#Canonical_plays). Do plays that are thematically similar to the one that you’re ranking appear among its most similar plays, according to cosine similarity? Another clue that you’re doing the right thing is if a play has a cosine of 1 with itself. If that’s not the case, then you’ve messed something up. Another good hint, is that there are a ton of plays about Henry. They’ll probably be similar to each other.

**Comparing Word Similarity using Term-Context Matrix**

Next, we will represent words as vectors in vector space. This will give us a way of representing some aspects of the *meaning* of words, by measuring the similarity of their vectors.

In our term-document matrix, the rows are word vectors. Instead of a |V| -dimensional vector, these row vectors only have D dimensions. Do you think that’s enough to represent the meaning of words? You can compute the most similar words for some words using the row vectors in your term-document matrix. Do the similar words make sense?

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 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.

Write 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.

You can now re-compute the most similar words for your test words using the row vectors in your term-context matrix instead of your term-document matrix. What is the dimensionality of your word vectors now? Do the most similar words make more sense than before?

**Weighting terms**

Your term-context matrix contains the raw frequency of the co-occurrence of two words in each cell. 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. You should implement two weighting schemes:

* Positive pointwise mutual information (PPMI) (implement create\_PPMI\_matrix)
* Term frequency inverse document frequency (tf-idf) (implement create\_tf\_idf\_matrix)

*Warning, calculating PPMI for your whole*|V|*-by-*|V|*matrix might be slow. You might want to find out other ways of using matrix operations to optimize performance. Once you compute the PPMI scores, you can save them in a file so that next time the grader runs your code, the program can just load these scores from the file.*

**Weighting terms**

There are several ways of computing the similarity between two vectors. In addition to writing a function to compute cosine similarity (compute\_cosine\_similarity), you should also write functions to [Jaccard similarity](https://en.wikipedia.org/wiki/Jaccard_index) (compute\_jaccard\_similarity) and [Dice similarity](https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient) (compute\_dice\_similarity).

**Your Tasks**

All of the following are function stubs in the python code. You just need to fill them out.

Create matrices: (20 points)

* fill out create\_term\_document\_matrix
* fill out create\_term\_context\_matrix
* fill out create\_PPMI\_matrix
* fill out compute\_tf\_idf\_matrix

Compute similarities: (10 points)

* fill out compute\_cosine\_similarity
* fill out compute\_jaccard\_similarity
* fill out compute\_dice\_similarity

Do some ranking: (10 points)

* fill out rank\_plays
* fill out rank\_words

**Discussion (20 points), bonus points: 5**

Play with different vector representations and different similarity functions. Does one combination appear to work better than another? Do any interesting patterns emerge? Discuss your findings in your write up. To qualify for the full credit for this section, you must discuss at least two interesting findings. A really interesting finding will get an extra 5 bonus points.

Some patterns you could touch upon:

* The fourth column of will\_play\_text.csv contains the name of the character who spoke each line. Using the methods described above, which characters are most similar? Least similar?
* Do the vector representations of [female characters](https://en.wikipedia.org/wiki/Category:Female_Shakespearean_characters) differ distinguishably from [male ones](https://en.wikipedia.org/wiki/Category:Male_Shakespearean_characters)?
* Shakespeare’s plays are traditionally classified into [comedies, histories, and tragedies](https://en.wikipedia.org/wiki/Shakespeare%27s_plays). Can you use these vector representations to cluster the plays? Can you find which play is central to each category?
* Etc.

**Deliverables**

Here are the deliverables that you will need to submit by sending it via email to [wijaya@bu.edu](mailto:wijaya@bu.edu) and [rxtan@bu.edu](mailto:rxtan@bu.edu) :

* writeup.pdf for your discussion of interesting findings
* code. (the code is in Python3)
* README file on how to run your code. **The grader must be able to run your code for your submission to be graded**.