Understanding Shakespeare with Math

This assignment is due before 11:59PM on Tuesday, February 12, 2019.

Vector Space Models : Assignment 3

In this assignment you will implement many of the things you learned in Chapter 6 of the textbook. If you haven't read it yet, now would be a good time to do that. We'll wait. Done? Great, let's move on.

We will provide a corpus of Shakespeare plays, which you will use to create a term-document matrix and a term-context matrix. You'll implement a selection of the weighting methods and similarity metrics defined in the textbook. Ultimately, your goal is to use the resulting vectors to measure how similar Shakespeare plays are to each other, and to find words that are used in a similar fashion. All (or almost all) of the code you write will be direct implementations of concepts and equations described in Chapter 6, sections 6.3-6.7.

All difficulties are easy when they are known.

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

- Skeleton python code
- Data - csv of the complete works of Shakespeare
- Data - vocab the complete works of Shakespeare
- Data - list of all plays in dataset

Term-Document Matrix

You will 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
crown	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.

In your code you will write a function to `create_term_document_matrix` . This will let you be the hit of your next dinner party by being able to answer trivia questions like *how many words did Shakespeare use?*, which may give us a hint to the answer to *[How many words did Shakespeare know?]* The table will also tell you how many words Shakespeare used only once. Did you know that there's a technical term for that? In corpus linguistics they are called *hapax legomena*, but I prefer the term *singleton*, because I don't like snooty Greek or Latin terms.

Comparing plays

The term-document matrix will also let us do cool things like 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 canon that maybe they weren't authored by Shakespeare after all.

Let's begin by considering the column representing each play. Each column is a $|V|$-dimensional vector. Let's use some math to define the similarity of these vectors. By far the most common similarity metric is the cosine of the angle between the vectors. The cosine similarity metric is defined in Section 6.3 of the textbook.

> 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:

$$\text{dot-product}(\vec{v}, \vec{w}) = \vec{v} \cdot \vec{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

> 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 vector length is defined as

$$|\vec{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

> 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, following from the definition of the dot product between two vectors $\vec{v}$ and $\vec{w}$ as:

$$\vec{v} \cdot \vec{w} = |\vec{v}||\vec{w}| cos\Theta$$

$$\frac{\vec{v} \cdot \vec{w}}{|\vec{v}||\vec{w}|} = cos\Theta$$

> The cosine similarity metric between two vectors $\vec{v}$ and $\vec{w}$ thus can be computed

$$cosine(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

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.

Please implement `compute_cosine_similarity`, and for each play in the corpus, score how similar each other play is to it. Which plays are the closet to each other in vector space (ignoring self similarity)? Which plays are the most distant from each other?

How do I know if my rankings are good?

First, read all of the plays. Then perform at least three of them. Now that you are a true thespian, you should have a good intuition for the central themes in the plays. Alternately, take a look at this grouping of Shakespeare's plays into Tragedies, Comedies and Histories. 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.

Measuring word similarity

Next, we're going to see how we can 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? Try it out. In the same way that you computed the similarity of the plays, you can compute the similarity of the words in the matrix. Pick some words and compute 10 words with the highest cosine similarity between their row vector representations. Are those 10 words good synonyms?

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 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.

For this part of the assignment, you should 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)
- Term frequency inverse document frequency (tf-idf)

These are defined in Section 6.2 of the textbook.

Warning, calculating PPMI for your whole $|V|$-by-$|V|$ matrix might be slow. Our intrepid TA's implementation for PPMI takes about 10 minutes to compute all values. She always writes perfectly optimized code on her first try. You may improve performance by using matrix operations a la MATLAB.

Weighting terms

There are several ways of computing the similarity between two vectors. In addition to writing a function to compute cosine similarity, you should also write functions to `compute_jaccard_similarity` and `compute_dice_similarity`. Check out section 6.3.1. of the textbook for the definitions of the Jaccard and Dice measures.

Your Tasks

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

Create matrices:

- 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:

- fill out `compute_cosine_similarity`
- fill out `compute_jaccard_similarity`
- fill out `compute_dice_similarity`

Do some ranking:

- fill out `rank_plays`
- fill out `rank_words`

Report

In the ranking tasks, play with different vector representations and different similarity functions. Does one combination appear to work better than another? Do any interesting patterns emerge? Include this discussion in your writeup.

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?
- Shakespeare's plays are traditionally classified into comedies, histories, and tragedies. Can you use these vector representations to cluster the plays?
- Do the vector representations of female characters differ distinguishably from male ones?

Here is one of the sample reports that illustrates what we are looking for. Here is another one.

Extra credit

Quantifying the goodness of one vector space representation over another can be very difficult to do. It might ultimately require testing how the different vector representations change the performance when used in a downstream task like question answering. A common way of quantifying the goodness of word vectors is to use them to compare the similarity of words with human similarity judgments, and then calculate the correlation of the two rankings.

If you would like extra credit on this assignment, you can quantify the goodness of each of the different vector space models that you produced (for instance by varying the size of the context window, picking PPMI or tf-idf, and selecting among cosine, Jaccard, and Dice). You can calculate their scores on the SimLex999 data set, and compute their correlation with human judgments using Kendall's Tau.

Add a section to your writeup explaining what experiments you ran, and which setting had the highest correlation with human judgments.

More Optional Fun Extra Credit options

So you've built some machinery that can measure similarity between words and documents. We gave you a Shakespeare corpus, but you can use any body of text you like. For example, check out Project Gutenberg for public domain texts. The sky's the limit on what you can do, but here are some ideas:

- *Term-Character Matrix*. Our data set.
- *Novel recommender system*. Maybe you enjoyed reading *Sense and Sensibility* and *War and Peace*. Can you suggest some similar novels? Or maybe you need some variety in your consumption. Find novels that are really different.
- *Other languages*. Do these techniques work in other languages? Project Gutenberg has texts in a variety of languages. Maybe you could use this to measure language similarity?
- *Modernizing Shakespeare*. When I read Shakespeare in high school, I had the dickens of a time trying to understand all the weird words in the play. Some people have re-written Shakespeare's plays into contemporary English. An awesome NLP researcher has compiled that data. Use her data and your vector space models to find contemporary words that mean similar things to the Shakespearean English.

Deliverables

Here are the deliverables that you will need to submit:

- writeup.pdf
- code (.zip). It should be written in Python 3.

Recommended readings

Vector Semantics. Dan Jurafsky and James H. Martin. Speech and Language Processing (3rd edition draft) .

From Frequency to Meaning: Vector Space Models of Semantics. Peter D. Turney and Patrick Pantel. Journal of Artificial Intelligence Research 2010. Abstract BibTex

Paraphrasing for Style. Wei Xu, Alan Ritter, Bill Dolan, Ralph Grisman, and Colin Cherry. Coling 2012. Abstract BibTex

Evaluation methods for unsupervised word embeddings. Tobias Schnabel, Igor Labutov, David Mimno, Thorsten Joachims. EMNLP 2015. Abstract BibTex

Community Evaluation and Exchange of Word Vectors at wordvectors.org. Manaal Faruqui and Chris Dyer. ACL demos 2014. Abstract BibTex