Topic Modeling with Singular Value Decomposition and Non-negative Matrix Factorization

Shu Xia <sxia38@wisc.edu>

Ziqian Deng <zdeng67@wisc.edu>

May 1, 2020

ECE/CS/ME 532 Spring 2020

**1 Abstract**

Natural Language Processing (NLP) is a rapidly advancing branch of unsupervised machine learning concerned with the ability of a computer to understand, analyze, manipulate, and potentially generate human language. Applications of NLP include Topic Modeling, where abstract "topics" that occur in a collection of documents need to be detected; Machine Translation, where text or speech are translated from one language to another; Sentiment Analysis, which allows businesses to identify customer sentiment toward products, brands or services in online conversations and feedback; Spam Filter, where unsolicited and unwanted email and messages are prevented from getting into user’ inbox, etc.

And just like in many other machine learning fields, matrix decomposition is one of the most important underlying methods in NLP. To help us start the study in NPL, this activity will introduce how to apply Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF) to conduct Topic Modeling on text data.

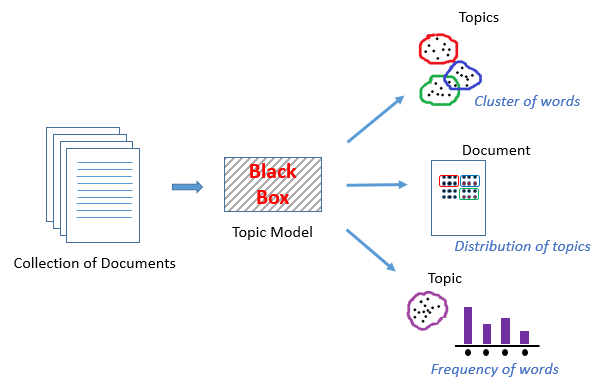
1.1 Learning objectives

* Have a knowledge of the wide range of applications in NLP
* Understand what is topic modeling, and learn how to represent topic modeling problems using matrix method.
* Learn to conduct basic text preprocessing (Tokenization, Stemming, Lemmatization, etc)
* Refresh their knowledge in SVD and apply SVD to topic modeling problems
* Learn the Non-negative Matrix Factorization (NMF) and apply NMF to topic modeling problems
* Compare these two matrix factorization techniques and explain the advantages of NMF

**2 Background**

**2.1 What is a topic model**

A Topic Model can be defined as an unsupervised technique to discover topics across various text documents. These topics are abstract in nature, i.e., words which are related to each other form a topic. Similarly, there can be multiple topics in an individual document. For the time being, let’s understand a topic model as a black box, as illustrated in the below figure:



This black box (topic model) forms clusters of similar and related words which are called topics. These topics have a certain distribution in a document, and every topic is defined by the proportion of different words it contains.

**2.2 Introduction to basic text preprocessing**

**2.2.1 Tokenization, Stemming, Lemmatization**

We will start by learning the basic text preprocessing techniques. Cleaning and preparation are crucial for many tasks, and NLP is no exception. Text preprocessing is usually the first step you’ll take when faced with an NLP task. Without preprocessing, our computer interprets "the", "The", and "<p>The" as entirely different words. There is a lot we can do here, depending on the formatting you need. Common tasks include:

Noise removal — stripping text of formatting (e.g., HTML tags, stop words).

***Tokenization*** — breaking text into individual words.

Normalization — cleaning text data in any other way:

* ***Stemming*** is a blunt axe to chop off word prefixes and suffixes. “booing” and “booed” become “boo”, but “sing” may become “s” and “sung” would remain “sung.”
* ***Lemmatization*** is a scalpel to bring words down to their root forms. For example, NLTK’s savvy lemmatizer knows “am” and “are” are related to “be.”

In Python, various packages will do most of the preprocessing tasks for us. Use the given code to try different methods to deal with text data.

**Warm-up Question 1:** try lemmatizing and stemming the following collections of words:

* fly, flies, flying
* organize, organizes, organizing
* universe, university

Did you find anything interesting? Did the meaning of the individual word change a lot after lemmatization and stemming?

**2.2.2 Represent text documents with matrix**

Next, we will learn to use the term-document matrix to represent the text data. A document-term matrix or term-document matrix is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. In a document-term matrix, rows correspond to documents in the collection and columns correspond to terms. For example:

Document 1 = “ the cat sat on my face”

Document 2 = “ the dog sat on my bed”

The simplest strategy is to create a vector of all possible words, and for each document count how many times each word appears. In this way, the mathematical matrix will be:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | bed | cat | dog | face | my | on | sat | the |
| Document 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| Document 2 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |

The problem with this counting strategy is that we use a lot of words commonly, that just don’t mean much. In fact, the most commonly used word in English “the” makes up 7% of the words we speak, which is double the frequency of the next most popular word “of”. So, if we construct our document matrix out of counts, then we end up with numbers that don’t contain much information, unless our goal is to see who uses “the” most often.

Rather than just counting, a technique in NLP known as ***term frequency-inverse document frequency (tf-idf)*** is better. TF-IDF score of a word is a numerical statistic for ranking its importance by deprioritizing the most common words and prioritizing less frequently used terms as topics.

The TF-IDF score of word, *w*, is:

Where

)

If we use the TF-IDF for the previous example, the matrix will become:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | bed | cat | dog | face | my | on | sat | the |
| Document 1 | 0 | 0.115525 | 0 | 115525 | 0 | 0 | 0 | 0 |
| Document 2 | 0.115525 | 0 | 0.115525 | 0 | 0 | 0 | 0 | 0 |

Compared with the previous matrix, this new matrix only gave scores to meaningful words such as “cat” and “face” for document 1 and “bed” and “dog” for document 2, while it gave 0 to those less important words appearing commonly and equally in both documents. Therefore, this matrix is easier for human and machine to interpret.

**Warm-up Question 2:** try different strategies to count words in three of Donald Trump’s tweets and turn them into a matrix.

1. Use the simple counting strategy first
2. Then complete the line of code that is unfinished and try the TF-IDF strategy

**2.3 Singular Value Decomposition and Latent Semantic Analysis**

Now we will dive into using Singular Value Decomposition (SVD) to address Topic Modeling. Sometimes we hear topic modeling as ***Latent Semantic Analysis (LSA)***which is an algorithm that leverages SVD to produce a set of concepts related to the documents and terms.

**2.3.1 Overview of Latent Semantic Analysis**

All languages have their own intricacies and nuances which are quite difficult for a machine to capture (sometimes they’re even misunderstood by us humans!). This can include different words that mean the same thing, and also the words which have the same spelling but different meanings.

For example, consider the following two sentences:

1. I liked his last novel quite a lot.
2. We would like to go for a novel marketing campaign.

In the first sentence, the word ‘novel’ refers to a book, and in the second sentence it means new or fresh.

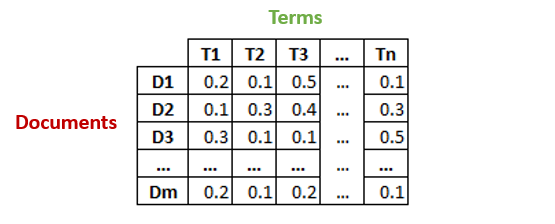
We can easily distinguish between these words because we are able to understand the context behind these words. However, a machine would not be able to capture this concept as it cannot understand the context in which the words have been used. This is where Latent Semantic Analysis (LSA) comes into play as it attempts to leverage the context around the words to capture the hidden concepts, also known as topics.

So, simply mapping words to documents won’t really help. What we really need is to figure out the hidden concepts or topics behind the words. LSA is one such technique that can find these hidden topics. Let’s now deep dive into the inner workings of LSA.

**2.3.2 Singular Value Decomposition in Latent Semantic Analysis**

Let’s say we have *m* number of text documents with *n* number of total unique terms (words). We wish to extract *k* topics from all the text data in the documents. The number of topics, k, has to be specified by the user. Here are the steps to conduct the LSA:

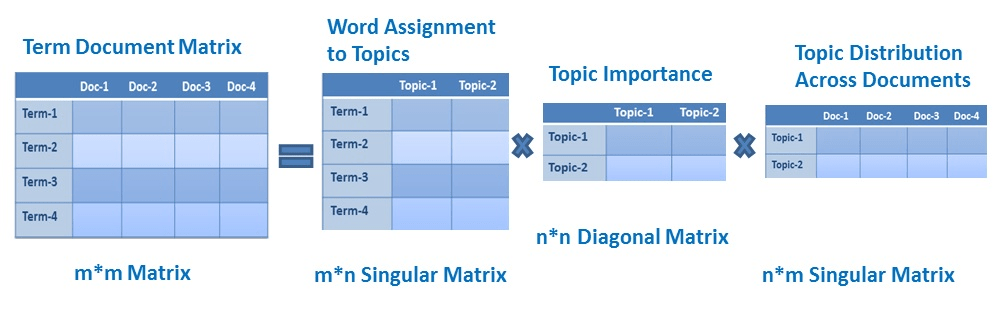
* Generate a document-term matrix of shape m x n having TF-IDF scores.



* Then, we will reduce the dimensions of the above matrix to *k* (number of desired topics) dimensions, using SVD.

We have learned SVD. A matrix A with rank r has a SVD of the form:

where U is a matrix with orthonormal columns, V is a matrix with orthonormal columns, and S is an diagonal matrix with the entries sorted in decreasing order. The entries of the S matrix are the singular values, and the U and V matrices are the left and right singular vectors, corresponding to term and document vectors. This is simply a re-representation of the A matrix using orthogonal indexing dimensions.



Specifically, LSA uses the truncated SVD, in which the k largest singular values are retained, and the remainder set to 0. The resulting representation is the best k-dimensional approximation to the original matrix in the least-squares sense. Each document and term is now represented as a k-dimensional vector in the space derived by the SVD..

**Warm-up Question 3:** In the previous question, we define our own functions to compute the TF-IDF scores, but actually, Python has various packages such as scikit-learn that can help us compute the TF-IDF quickly. Please follow the provided code and learn to conduct LSA with SVD on a text data *raw\_forum\_posts.dat* which contains posts from a data science forum.

* After printing out the topics and their related terms, could you summarize the topics using your own words? Try to use a sentence or a phrase to summarize topic 3, 4 and 5. For example, the topic 0 could be summed up as “using big data”.

**2.4 Non-negative Matrix Factorization**

**3 Main Activity**

**4 References**

**5 Appendix**