Practical 1Vector Space Model – Building the simplest search engine

Course Instructor: Shailesh B. Pandey

B.E. Computer (Seventh Semester)

Everest Engineering College

Fall-2022

1. Introduction

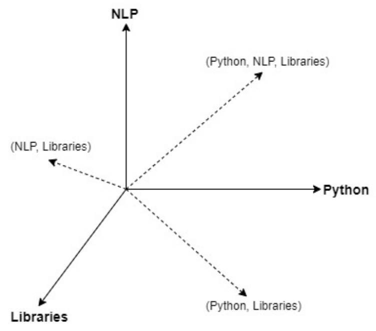
In this lab we will explore the notion of a *document vector* which captures the importance of the terms (words, for this lab) in the document. The documents can thus be plotted as vectors in a graph and we can calculate their length and talk about distance between two vectors. The representation of documents as vectors in some vector space is known as the *vector space model*.

The simplest way is to represent a document by the terms (words) contained in it. The set of documents thus creates a vector space where there is one axis for each term. The obvious limitation of such a representation is that we lose the order in which the words appear in the document i.e. context. Therefore, such a representation is known as a *bag-of-words* representation.

To construct the vector space for words we assume that there is no relationship between the words. Speaking mathematically, we assume that the words are linearly independent basis vectors. Basically, what this means is that each word represents an axis in the vector space. We now have the space where we can place the document vectors. As you can imagine this is going to be a very high dimensional space. The size of vocabulary of the Google News Corpus is 3 million. That means we have 3 million axes if we consider all the words.

As an example let us consider that vocabulary is limited to three words: Python, NLP and Libraries. Each word is represented as an axis. A document either contains one or two or all of these three words. In the table below Document A (DocA) contains all three words and thus its vector representation is (1,1,1).

	Python	**NLP**	**Libraries**
DocA	1	1	1
DocB	1	0	1
DocC	0	1	1



In general, we do not use the binary representation. It is common to use the term-frequency (TF) and inverse document frequency (IDF). Once we have defined word representations and document vectors, we want some way to measure relationship between the documents. We often talk about distance and similarity. We can ask the question: How similar are Document A and Document B? It is not useful to use the magnitude of the difference between the two vectors. This is because two documents containing the same terms can appear to be very different just because one document is much longer. Instead, one of the most commonly used measure is *cosine similarity*.

$$similarity = \cos\theta = \frac{\mathbf{A}.\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Term Frequency and Inverse Document Frequency

We want to calculate similarity based on the weight of the terms in the documents. The simplest approach is to assign the number of occurrences as weight of the term for the document. Using occurrence as weight is called the *term frequency*. Term frequency (TF) of a term *t* in document *d* is denoted by $tf_{t,d}$. This is the *bag-of-words* scheme as we are considering the occurrences but not the ordering.

Term frequencies alone do not have much discriminative power. For instance, the word 'the' appears a lot in documents written in English. The effect of such words must be attenuated when we calculate similarities. This is where Inverse Document Frequency (IDF) comes in the picture.

$$idf_t = \log\frac{N}{df_t}$$

Here N is the number of documents in the collection. df_t is the document frequency of the term *t*. It is the number of documents in the collection that contains the term.

2. Tasks

In this lab you will: (a) implement the tf-idf weighing scheme, and (b) find documents relevant (similar) to the query. You are given three very short documents containing a sentence each. Doc1 = "The sky is blue", Doc2 = "The sun is bright", Doc3 = "The sun in the sky is bright".

Task A:

1. Write a method/function tf() which takes a document and a term as an argument. The function returns the occurrence count of the term in that document. [*Note: It is often useful to divide the count by number of words in the document. Could you see the reason why this is useful?*]
2. Write a function idf() which takes the document collection (in our case the list/array) and the term as an argument. The function returns the idf score for the term.
3. Write a function tf-Idf() which multiplies the tf and idf scores from the earlier functions for a term.

Task B:

Your task is to construct the document vectors for each document in the collection. To do so you identify the vocabular of the document collection. Vocabulary is nothing but unique words in the documents. In this case the vocabulary is [the, sky, is, blue, sun, bright, in]. Notice that I have ignored case as in "The" and "the". They are treated as the same word.

To create the vector for Doc1 we simply calculate tf-idf scores for each word in the vocabulary for Doc1. The vector is: v1 = {`the=0.0, sky=0.10, is=0.0, blue=0.27, sun=0.0, bright=0.0, in=0.0`}.

Write answer to these:

1. Why is the tf-idf score for the term "bright" zero for Doc1?
2. Why is the tf-idf score for the term "the" zero for Doc1?

3. What can we tell from seeing that tf-idf score for the term “blue” is larger than “sky”?

Task C:

1. Write a function cosineSimilarity(d_1, d_2) that calculates cosine similarity between two document vectors passed as arguments.
2. Write a function score(q,d) that calculates score for a query (a search string like what you give to google search) in document d. You will implement this formula:

$$score(q,d) = \sum_{t \in q} tf - idf_{t,d}$$

Task D:

1. Right now, the documents are list of strings. Re-write the code to construct the lists from a document. That is, open three separate files and read its contents into doc1, doc2 and doc3 respectively. Construct doc1 from sports related news, doc2 from entertainment related news and doc3 from business related news.

3. Report Format

1. Cover Page
2. Introduction: Short summary of what you are doing in this lab – one to two paragraphs.
3. Summary: Short summary of what you learned by solving this practical, what topics/ideas/equations/algorithms introduced in the lab you found difficult to understand and implement – one or two paragraphs.
4. Solution: Task-wise solution. There are four tasks: Task A to Task D.
5. Discussion: Discussions on what you found after you ran the program and saw the results. Write what did you observe.
6. Appendix: Pretty-formatted and commented code.

4. Submission guidelines

- You are required to complete the tasks and submit a report with pretty-formatted and well-commented code. Certain marks are allotted for clarity and readability.
- You are supposed to do the exercises on your own. If you need help you can ask me or the lab instructor for the course. You are not supposed to copy your friend’s code. It is very unlikely that even for the same method implementation two people will write identical code. It will be caught.
- Each submission will have a viva-voce. If you have done the exercises by yourself, you will have no problem at all.

5. Programming hints

You can store the document as an *array (of strings) or a list (of strings)* for the purpose of this lab. Once you represent a document as a list, you can think of the document collection as a list of lists (of strings).

Here is the code to do this:

```java
import java.util.Arrays;
import java.util.List;

public class TermFrequency {

	public void printList(List<String> doc) {
		for (String word : doc) {
			System.out.print(word+" ");
		}
		System.out.print("\n");
	}

	public void printListofLists(List<List<String>> docs) {
		for (List<String> doc : docs) {
			for (String word : doc) {
				System.out.print(word+" ");
			}
			System.out.print("\n");
		}
	}

	public static void main(String[] args) {
		TermFrequency freq = new TermFrequency();
		List<String> doc1 = Arrays.asList("the","sky", "is", "blue");
		List<String> doc2 = Arrays.asList("the","sun","is","bright");
		List<String> doc3 =
		Arrays.asList("the","sun","in","the","sky","is","bright");
		List<List<String>> corpus = Arrays.asList(doc1, doc2, doc3);

		freq.printList(doc1);
		freq.printList(doc2);

		System.out.println("\nDocument Collection:");
		freq.printListofLists(corpus);
	}
}
```