Document-Term Matrix in NLP: *Count and TF-IDF Scores Explained*

Natural Language Processing (NLP) is a subfield of Linguistics and Computer Science that deals with processing and understanding huge volumes of unstructured text data.

To leverage NLP to analyze textual data, you'll have to come up with a numeric representation of the text corpus, or the collection of documents. This tutorial will teach you all about **Document-Term Matrix (DTM)**—a matrix representation of the text corpus. By the end of this tutorial, you'll have learned enough to understand the Document-Term Matrix representation, and the significance of the TF-IDF score. And you'll also be able to generate Document-Term matrices for any text corpus in scikit-learn.

Table of Contents

What is the Document-Term Matrix?

In the context of NLP, a text corpus is simply a collection of documents. And a document can be a sentence, a group of sentences, or even a phrase—depending upon the use case.

Here's a simple example of a corpus with only 3 documents

`D1`

,

`D2`

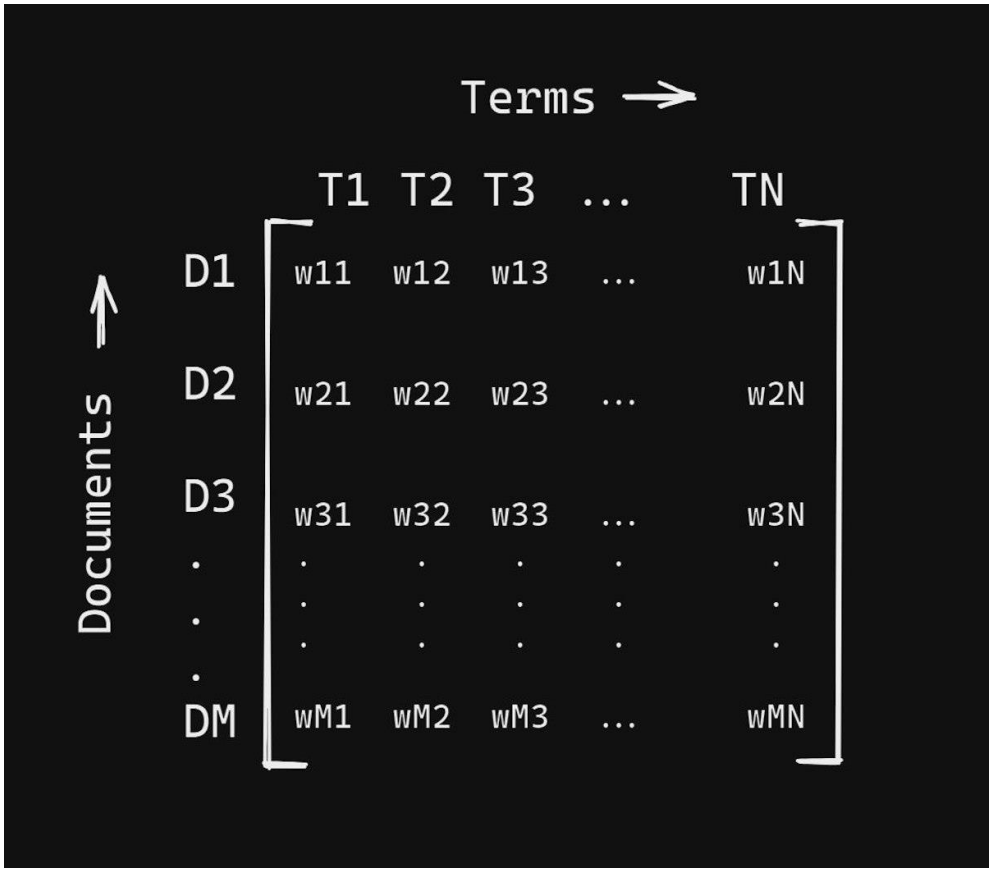
and

`D3`

.

```
D1: NLP is super cool to learn and apply.
D2: I am currently learning NLP, you can too!
D3: CS224n at Stanford is the best NLP class you can ever take!
```

Let's suppose the text corpus contains M documents, and N terms in all. The general structure of the Document-Term Matrix is shown below:



Document-Term Matrix

Let's parse the matrix representation:

- `D1`, `D2`, ..., `DM` are the `M` documents
- `T1`, `T2`, ..., `TN` are the `N` terms.

One of the simplest ways of populating the Document-Term Matrix is using the number of occurrences of the `N` terms across all the `M` documents.

- The entry `w11` denotes the number of times the term `T1` occurs in the document `D1`, `w12` denotes the number of times the term `T2` occurs in the document `D1`, and so on.
- In general, `wij` denotes the number of times the term `Tj` occurs in the document `Di`.

This way, a term that occurs *frequently* in the text corpus will have a much *higher* score, and a term that occurs *rarely* in the text corpus will have a much *lower* score. The ordering and semantic relationship between the terms is lost.
Populating the Document-Term Matrix as shown above—with the number of times a term occurs in the documents is both simple and intuitive. However, are there any caveats you should be aware of?

Head over to the next section to find out.

What is the TF-IDF Score?

In populating the Document-Term Matrix with the number of occurrences, frequently occurring terms are assigned a higher score than the rarely occurring terms.
Now, take a step back, and ask yourself the question: "Is a frequently occurring term necessarily important?"

Well, this may not always be the case.

For example, when you're reading a piece of text on astronomy—you'll come across words like ***'sky'*** and ***'stars'*** very often. But would you not like to see beyond these words to understand what the text is talking about?

This is exactly the motivation behind TF-IDF score—another useful metric that you can use to populate the Document-Term Matrix.

TF-IDF score is a combination of two metrics: the **Term Frequency (TF)** and the **Inverse Document Frequency (IDF)**.

TF-IDF Score Equation

The TF-IDF score is given by the following equation:

$$w_{ij} = TF_{ij} * log\left(\frac{M}{df_j}\right)$$

Where,

- `TF_ij` is the number of times the term `Tj` occurs in the document `Di`
- `Dfj` is the number of documents containing the term `Tj`

Let's parse the above expression:

- The first term, `TF_ij` is called the **Term Frequency (TF)**. It is identical to the count we had in the previous section: the number of times the term `Tj` occurs in the document `Di`

  So, the more frequently a term occurs in a particular document, the higher its TF score.
- The second term is the **Inverse Document Frequency (IDF)**.

$$IDF = log\left(\frac{M}{df_j}\right)$$

Here,

• `M` is the total number of documents, and

• `df_j` is the number of documents containing the term `TF_ij`

What is the Significance of the TF-IDF Score?

Let's first understand the significance of the IDF score.

If the term
`Tj` occurs in all the documents, the value of
`df_j = M`
. This would leave you with
`log(M/M) = log(1)`

Recall from your school math: log(1) = 0

On the other hand, if the term
`Tj` occurs in *only one* of the
`M` documents, you'll have
`log(M/1) = log(M)`

Notice how this is the maximum value that the IDF score can take.
Putting it all together:

- A term that occurs frequently in a particular document has a higher TF score.
- A term that occurs rarely across the entire corpus has a higher IDF score.
- The TF-IDF score is the product of the TF and the IDF scores.

In simple terms, the TF-IDF score says, "Hey! I believe terms that occur frequently in a particular document are likely more important than terms that occur frequently in all documents in the corpus!"

Though the TF-IDF score fails to capture contextual relationships between terms in the corpus, it's still considered an effective metric and is preferred over counting only the number of occurrences.

Now that you've learned the basics of Document-Term Matrix and the TF-IDF scores, let's now see how you can generate the DTM and populate it with the TF-IDF score in scikit-learn.

How to Generate the Document-Term Matrix in scikit-learn

1. First step is to import the `TfidfVectorizer` class from scikit-learn's feature extraction module for text data. This `TfidfVectorizer` helps generate the Document-Term Matrix populated with the TF-IDF score.

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

2. Next, get our text corpus
   The following code cell contains a piece of text on computer programming—and this will be our corpus of interest.

```
text=["Computer programming is the process of designing and building an executable computer program to accomplish a specific computing result or to perform a specific task.",
      "Programming involves tasks such as: analysis, generating algorithms, profiling algorithms' accuracy and resource consumption, and the implementation of algorithms in a chosen programming language (commonly referred to as coding).",
      "The source code of a program is written in one or more languages that are intelligible to programmers, rather than machine code, which is directly executed by the central processing unit.",
      "The purpose of programming is to find a sequence of instructions that will automate the performance of a task (which can be as complex as an operating system) on a computer, often for solving a given problem.",
      "Proficient programming thus often requires expertise in several different subjects, including knowledge of the application domain, specialized algorithms, and formal logic.",
      "Tasks accompanying and related to programming include: testing, debugging, source code maintenance, implementation of build systems, and management of derived artifacts, such as the machine code of computer programs.",
      "These might be considered part of the programming process, but often the term software development is used for this larger process with the term programming, implementation, or coding reserved for the actual writing of code.",
      "Software engineering combines engineering techniques with software development practices.",
      "Reverse engineering is a related process used by designers, analysts and programmers to understand and re-create/re-implement"]
```

3. Let's instantiate a `Tfidfvectorizer` object. Let's call it `vectorizer`

```
vectorizer = TfidfVectorizer(stop_words='english',smooth_idf=True)
```

   Now, call the `fit_transform()` method on the `vectorizer` with `text` as the argument.

```
input_matrix = vectorizer.fit_transform(text)

print(input_matrix)
 # [truncated view]
 # sparse representation of the DTM
 # non-zero indices and TF-IDF score
  (0, 83)    0.21527315443847012
  (0, 55)    0.25487697925239533
  (0, 73)    0.25487697925239533
  (0, 20)    0.25487697925239533
  (0, 80)    0.5097539585047907
  (0, 1)     0.25487697925239533
  (0, 63)    0.21527315443847012
  (0, 33)    0.25487697925239533
  (0, 11)    0.25487697925239533
  (0, 27)    0.25487697925239533
```

As you can see, the `input_matrix` has the scores, and the indices corresponding to the non-zero entries only—and not the entire matrix.

To cast it into a matrix of dimensions `M x N`, you can use the `to_dense()` method, as shown in the code snippet below:

```
input_matrix = vectorizer.fit_transform(text).todense()
# Truncated view of the entire matrix

[[0.         0.25487698 0.         0.         0.         0.
  0.         0.         0.         0.         0.25487698
  0.         0.         0.         0.         0.         0.
  0.         0.37434759 0.25487698 0.         0.         0.
  0.         0.         0.         0.25487698 0.         0.
  0.         0.         0.         0.25487698 0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.25487698 0.         0.         0.         0.18717379
  0.         0.         0.         0.21527315 0.         0.13251329
  0.         0.         0.         0.         0.         0.
  0.         0.25487698 0.         0.         0.         0.
  0.         0.         0.50975396 0.         0.         0.21527315
  0.         0.         0.         0.         0.         0.
  0.         0.         0.        ]
```

```
 [0.         0.         0.22103745 0.         0.56007525
0.22103745
  0.         0.         0.         0.         0.
  0.         0.22103745 0.         0.18669175 0.         0.22103745
  0.         0.         0.         0.         0.22103745 0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.22103745 0.         0.         0.16232309 0.
  0.         0.         0.         0.22103745 0.         0.22103745
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.22103745 0.         0.         0.22983952
  0.         0.         0.22103745 0.         0.         0.
  0.22103745 0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.18669175 0.         0.         0.         0.         0.
  0.         0.         0.        ]
```

Conclusion

I hope this tutorial helped you understand what the Document-Term Matrix (DTM) is, and the significance of the TF-IDF score.
You can use the above code to generate the DTM for your favorite piece of text—simply replace the `text`

with your text corpus, and you're all ready to go!

Generating the DTM is often the first step in many unsupervised NLP tasks such as topic modeling.