Published in Towards Data Science

Cory Maklin Follow

May 5, 2019 · 4 min read ·

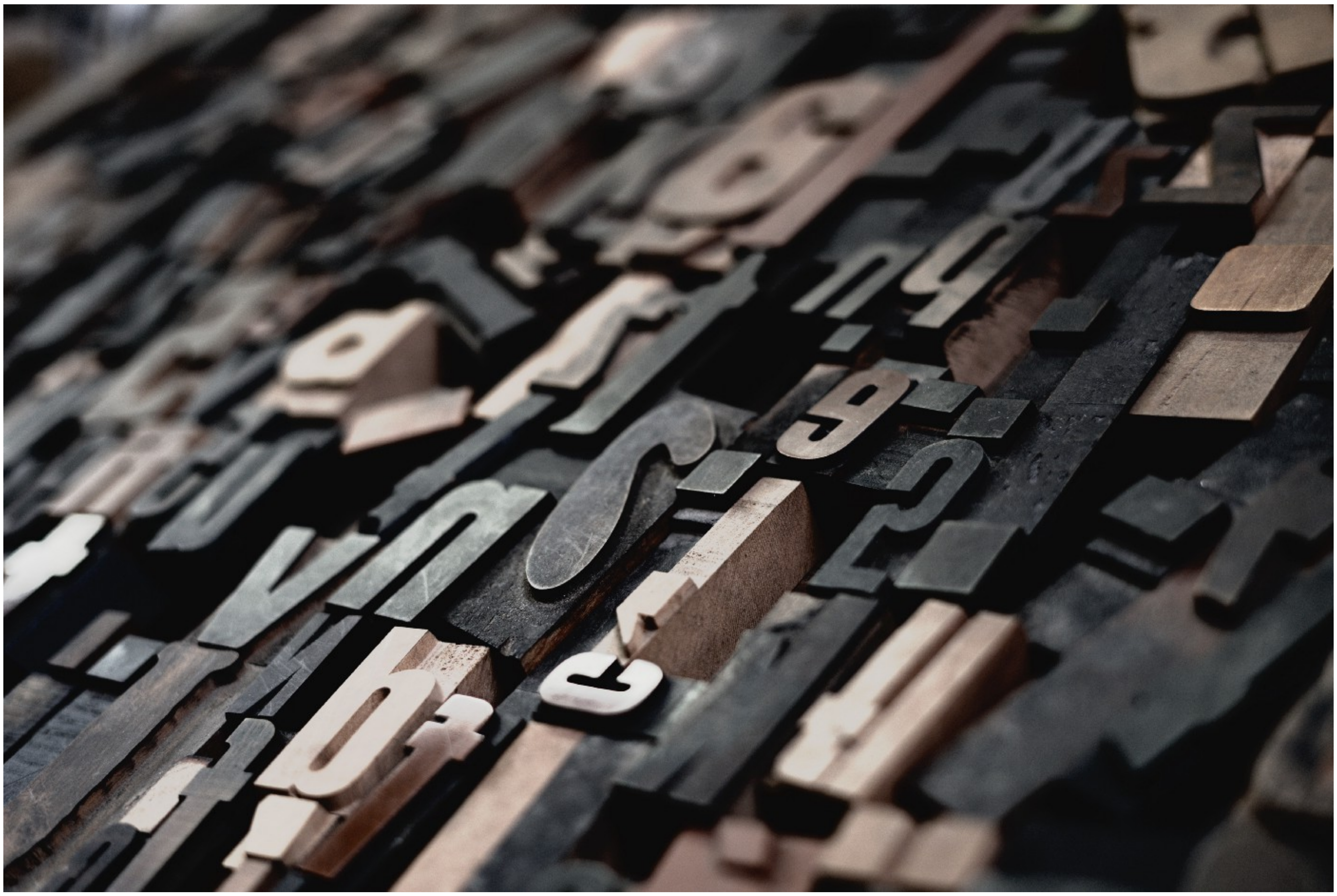

Photo by Raphael Schaller on Unsplash

TF IDF | TFIDF Python Example

Natural Language Processing (NLP) is a sub-field of artificial intelligence that deals understanding and processing human language. In light of new advancements in machine learning, many organizations have begun applying natural language

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
```

In this article, we'll be working with two simple documents containing one sentence each.

```
documentA = 'the man went out for a walk'
documentB = 'the children sat around the fire'
```

Machine learning algorithms cannot work with raw text directly. Rather, the text must be converted into vectors of numbers. In natural language processing, a common technique for extracting features from text is to place all of the words that occur in the text in a bucket. This aproach is called a **bag of words** model or **BoW** for short. It's referred to as a *"bag"* of words because any information about the structure of the sentence is lost.

```
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
```

```
['the', 'man', 'went', 'out', 'for', 'a', 'walk']
```

By casting the bag of words to a set, we can automatically remove any duplicate words.

```
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
```

Open in app

```
for word in bagOfWordsA:
    numOfWordsA[word] += 1

numOfWordsB = dict.fromkeys(uniqueWords, 0)

for word in bagOfWordsB:
    numOfWordsB[word] += 1
```

	a	around	children	fire	for	man	out	sat	the	walk	went
0	1	0	0	0	1	1	1	0	1	1	1
1	0	1	1	1	0	0	0	1	2	0	0

Another problem with the bag of words approach is that it doesn't account for noise. In other words, certain words are used to formulate sentences but do not add any semantic meaning to the text. For example, the most commonly used word in the english language is *the* which represents 7% of all words written or spoken. You couldn't make deduce anything about a text given the fact that it contains the word **the**. On the other hand, words like **good** and **awesome** could be used to determine whether a rating was positive or not.

In natural language processing, useless words are referred to as stop words. The python **natural language toolkit** library provides a list of english stop words.

```
from nltk.corpus import stopwords

stopwords.words('english')
```

{'ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'there', 'about', 'once', 'during', 'out', 'very', 'having', 'with', 'they', 'own', 'an', 'be', 'some', 'for', 'do', 'its', 'yours', 'such', 'into', 'of', 'most', 'itself', 'other', 'off', 'is', 's', 'am', 'or', 'who', 'as', 'from', 'him', 'each', 'the', 'themselves', 'until', 'below', 'are', 'we', 'these', 'your', 'his', 'through', 'don', 'nor', 'me', 'were', 'her', 'more', 'himself', 'this', 'down', 'should', 'our', 'their', 'while', 'above', 'both', 'up', 'to', 'ours', 'had', 'she', 'all', 'no', 'when', 'at', 'any', 'before', 'them', 'same', 'and', 'been', 'have', 'in', 'will', 'on', 'does', 'yourselves', 'then', 'that', 'because', 'what', 'over', 'why', 'so', 'can', 'did', 'not', 'now', 'under', 'he', 'you', 'herself', 'has', 'just', 'where', 'too', 'only', 'myself', 'which', 'those', 'i', 'after', 'few', 'whom', 't', 'being', 'if', 'theirs', 'my', 'against', 'a', 'by', 'doing', 'it',

Open in app

words using TF-IDF.

Term Frequency (TF)

The number of times a word appears in a document divded by the total number of words in the document. Every document has its own term frequency.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

The following code implements term frequency in python.

```
def computeTF(wordDict, bagOfWords):
    tfDict = {}
    bagOfWordsCount = len(bagOfWords)
    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)
    return tfDict
```

The following lines compute the term frequency for each of our documents.

```
tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)
```

Inverse Data Frequency (IDF)

Open in app

documents in the corpus.

$$idf(w) = log(\frac{N}{df_t})$$

The following code implements inverse data frequency in python.

```
def computeIDF(documents):
    import math
    N = len(documents)

    idfDict = dict.fromkeys(documents[0].keys(), 0)
    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val))
    return idfDict
```

The IDF is computed once for all documents.

```
idfs = computeIDF([numOfWordsA, numOfWordsB])
```

Lastly, the TF-IDF is simply the TF multiplied by IDF.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{\;}{df_i}\right)$$

```
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf
```

Finally, we can compute the TF-IDF scores for all the words in the corpus.

```
tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)

df = pd.DataFrame([tfidfA, tfidfB])
```

	a	around	children	fire	for	man	out	sat	the	walk	went
0	0.099021	0.000000	0.000000	0.000000	0.099021	0.099021	0.099021	0.000000	0.0	0.099021	0.099021
1	0.000000	0.115525	0.115525	0.115525	0.000000	0.000000	0.000000	0.115525	0.0	0.000000	0.000000

Rather than manually implementing TF-IDF ourselves, we could use the class provided by sklearn.

```
vectorizer = TfidfVectorizer()

vectors = vectorizer.fit_transform([documentA, documentB])

feature_names = vectorizer.get_feature_names()

dense = vectors.todense()
```

	around	children	fire	for	man	out	sat	the	walk	went
0	0.000000	0.000000	0.000000	0.42616	0.42616	0.42616	0.000000	0.303216	0.42616	0.42616
1	0.407401	0.407401	0.407401	0.00000	0.00000	0.00000	0.407401	0.579739	0.00000	0.00000

The values differ slightly because sklearn uses a smoothed version idf and various other little optimizations. In an example with more text, the score for the word ***the*** would be greatly reduced.