Bag-of-words vs TFIDF vectorization –A Hands-on Tutorial

Home

Mimi Dutta — July 14, 2021

Beginner Machine Learning NLP python Text

This article was published as a part of the Data Science Blogathon

Whenever we apply any algorithm to textual data, we need to convert the text to a numeric form. Hence, there arises a need for some pre-processing techniques that can convert our text to numbers. Both bag-of-words (BOW) and TFIDF are pre-processing techniques that can generate a numeric form from an input text.

Bag-of-Words:

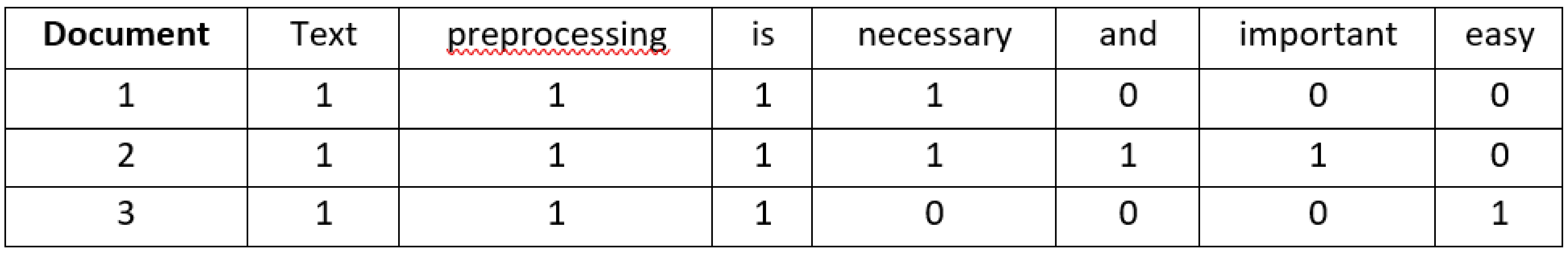
The bag-of-words model converts text into fixed-length vectors by counting how many times each word appears.

Let us illustrate this with an example. Consider that we have the following sentences:

- Text processing is necessary.
- Text processing is necessary and important.
- Text processing is easy.

We will refer to each of the above sentences as documents. If we take out the unique words in all these sentences, the vocabulary will consist of these 7 words: {'Text', 'processing', 'is', 'necessary', 'and', 'important, 'easy'}.

To carry out bag-of-words, we will simply have to count the number of times each word appears in each of the documents.



Document	Text	preprocessing	is	necessary	and	important	easy
1	1	1	1	1	0	0	0
2	1	1	1	1	1	1	0
3	1	1	1	0	0	0	1

Hence, we have the following vectors for each of the documents of fixed length -7:

Document 1: [1,1,1,1,0,0,0]

Document 2: [1,1,1,1,1,1,0]

Document 3: [1,1,1,0,0,0,1]

Limitations of Bag-of-Words:

- If we deploy bag-of-words to generate vectors for large documents, the vectors would be of large sizes and would also have too many null values leading to the creation of sparse vectors.
- Bag-of-words does not bring in any information on the meaning of the text. For example, if we consider these two sentences – "Text processing is easy but tedious." and "Text processing is tedious but easy." – a bag-of-words model would create the same vectors for both of them, even though they have different meanings.

Bag-of-words vs TFIDF vectorization –A Hands-on Tutorial

TFIDF works by proportionally increasing the number of times a word appears in the document but is counterbalanced by the number of documents in which it is present. Hence, words like 'this', 'are' etc., that are commonly present in all the documents are not given a very high rank. However, a word that is present too many times in a few of the documents will be given a higher rank as it might be indicative of the context of the document.

Term Frequency:

Term frequency is defined as the number of times a word (i) appears in a document (j) divided by the total number of words in the document.

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

Inverse Document Frequency:

Inverse document frequency refers to the log of the total number of documents divided by the number of documents that contain the word. The logarithm is added to dampen the importance of a very high value of IDF.

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

TFIDF is computed by multiplying the term frequency with the inverse document frequency.

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

Let us now see an illustration of TFIDF in the following sentences, that we refer to as documents.

Document 1: Text processing is necessary.

Document 2: Text processing is necessary and important.

Word	TF		IDF	TFIDF	
	Doc 1	Doc 2		Doc 1	Doc 2
Text	1/4	1/6	log (2/2) = 0	0	0
Processing	1/4	1/6	log (2/2) =0	0	0
Is	1/4	1/6	log (2/2) =0	0	0
Necessary	1/4	1/6	log (2/2) =0	0	0
And	0/4	1/6	log (2/1) =0.3	0	0.05
Important	0/4	1/6	log (2/1) =0.3	0	0.05

The above table shows how the TFIDF of some words are zero and some words are non-zero depending on their frequency in the document and across all documents.

The limitation of TFIDF is again that this vectorization doesn't help in bringing in the contextual meaning of the words as it is just based on the frequency.

Bag-of-words and TFIDF in Python

We can easily carry out bag-of-words or count vectorization and TFIDF vectorization using the sklearn library.

Bag-of-words vs TFIDF vectorization –A Hands-on Tutorial

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = ['Text processing is necessary.', 'Text processing is necessary and important.', 'Text processing is
easy.']
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
print(vectorizer.get_feature_names())
```

Printing the features gives the following output:

```
['and', 'easy', 'important', 'is', 'necessary', 'processing', 'text']
```

```
print(X.toarray())
```

```
[[0 0 0 1 1 1 1]
 [1 0 1 1 1 1 1]
 [0 1 0 1 0 1 1]]
```

The above array represents the vectors created for our 3 documents using the bag-of-words vectorization.

TFIDF Vectorization

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)
print(X.toarray())
```

```
[[0.         0.         0.         0.46333427 0.59662724 0.46333427
  0.46333427]
 [0.52523431 0.         0.52523431 0.31021184 0.39945423 0.31021184
  0.31021184]
 [0.         0.69903033 0.         0.41285857 0.         0.41285857
  0.41285857]]
```

The above array represents the vectors created for our 3 documents using the TFIDF vectorization.

***Important parameters to know** – Sklearn's CountVectorizer & TFIDF vectorization:*

- **max_features:** This parameter enables using only the 'n' most frequent words as features instead of all the words. An integer can be passed for this parameter.
- **stop_words:** You could remove the extremely common words like 'this', 'is', 'are' etc by using this parameter as the common words add little value to the model. We can set the parameter to 'english' to use a built-in list. We can also set this parameter to a custom list.
- **analyzer:** This parameter tells the model if the feature should be made of word n-grams or character n-grams. We can set it to be 'word', 'char' or 'char_wb'. Option 'char_wb' creates character n-grams only from text inside word boundaries.
- **ngram_range:** An n-gram is a string of words in a row. For example, in the sentence – "Text processing is easy.", 2-grams could be 'Text processing', 'processing is' or 'is easy'. We can set the ngram_range to be (x,y) where x is the minimum and y is the maximum size of the n-grams we want to include in the features. The default ngram_range is (1,1).
- **min_df, max_df:** These refer to the minimum and maximum document frequency that a word/n-gram should have to be used as a feature. The frequency here refers to the proportion of documents. Both the parameters have to be set in the range of [0,1].

About Author

Nibedita Dutta

Nibedita completed her master's in Chemical Engineering from IIT Kharagpur in 2014 and is currently working as a Senior Consultant at AbsolutData Analytics. In her current capacity, she works on building AI/ML-based solutions for clients from an array of industries.