Bag-of-Words

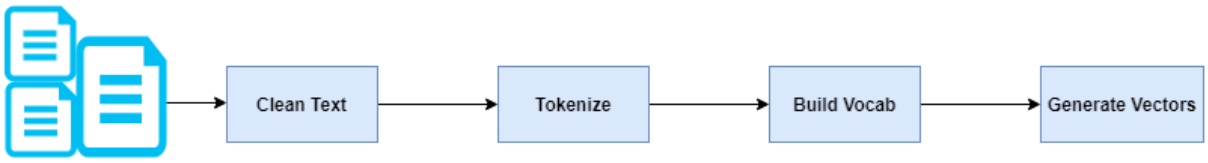
The **bag-of-words (BOW)** model is a representation that turns arbitary text into **fixed-length-vectors by counting how many times each word appears**.

This process is often referred to as **Vectorization**.

BOW is an approach widely used with:

- Natural language processing
- Information retrieval from documents
- Document classifications

On a high level, it involves the below steps:



Let's understand this with an example. Suppose we wanted to vectorize the following:

- the cat sat
- the cat sat in the hat
- the cat with the hat

We'll refer to each of these as a text **document**.

Each document will undergo several pre-processing steps like

- Case Conversion – converting all the text to lowercase
- Tokenization – breaking each document into words
- Removing HTML tags & noices
- Removal of accented characters
- Getting rid of special characters, numbers and symbols
- Expanding contractions – "couldn't" will be expanded as "could not"
- Stemming/Lemmatization – convert the words to it's root form
- Stopword removal

The pre-processing steps will ensure that the vocabulary that we make are meaningful ones.

At the end of the pre-processing step, we obtain a vocabulary list. I will make this as **Step 1**

Step 1: Determine the Vocabulary

We first define our vocabulary, which is the set of all words found in our document set. The only words that are found in the 3 documents above are: **the, cat, sat, in, the, hat, and with**.

Step 2: Count
To vectorize our documents, all we have to do is count how many times each word appears:

Document	the	cat	sat	in	hat	with
the cat sat	1	1	1	0	0	0
the cat sat in the hat	2	1	1	1	1	0
the cat with the hat	2	1	0	0	1	1

To obtain the above table, we use "Count Vectorizer". If we don't look for frequency for each word, then we can use either "get_dummies" or "one-hot encoder" or "numpy".

Now we have length-6 vectors for each document!

- the cat sat: [1, 1, 1, 0, 0, 0]
- the cat sat in the hat: [2, 1, 1, 1, 1, 0]
- the cat with the hat: [2, 1, 0, 0, 1, 1]

In this process (using BOW), we lose **contextual information**, e.g. where in the document the word appeared. t's like a literal **bag-of-words**: it only tells you *what* words occur in the document, not *where* they occurred.

Implementing BOW in Python

As I mentioned earlier, there are many ways to implement bag-of-words. In the below example, I will use keras Tokenizer.

```
from keras.preprocessing.text import Tokenizer

docs = [
  'the cat sat',
  'the cat sat in the hat',
  'the cat with the hat',
]

## Step 1: Determine the Vocabulary
tokenizer = Tokenizer()
tokenizer.fit_on_texts(docs)
print(f'Vocabulary: {list(tokenizer.word_index.keys())}')

## Step 2: Count
vectors = tokenizer.texts_to_matrix(docs, mode='count')
print(vectors)
```

The above code returns

```
Vocabulary: ['the', 'cat', 'sat', 'hat', 'in', 'with']
[[0. 1. 1. 1. 0. 0. 0.]
 [0. 2. 1. 1. 1. 1. 0.]
 [0. 2. 1. 0. 1. 0. 1.]]
```

Notice that the vectors here have length 7 instead of 6 because of the extra 0 element at the beginning. This is an inconsequential detail - Keras reserves index 0 and never assigns it to any word.

How is BOW useful?

Despite being a relatively basic model, BOW is often used for **Natural Language Processing (NLP)** tasks like Text Classification. Its strengths lie in its simplicity: it's inexpensive to compute, and sometimes simpler is better when positioning or contextual info aren't relevant.