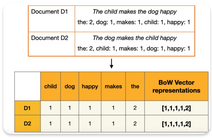

Document D1	*The child makes the dog happy* the: 2, dog: 1, makes: 1, child: 1, happy: 1
Document D2	*The dog makes the child happy* the: 2, child: 1, makes: 1, dog: 1, happy: 1

↓

	child	dog	happy	makes	the	BoW Vector representations
D1	1	1	1	1	2	[1,1,1,1,2]
D2	1	1	1	1	2	[1,1,1,1,2]

Def: a method for numerically representing text data that disregards the order of words and focuses solely on the frequency of their occurrence

Process of Bag of Words (BoW):

Tokenization:
- split into individual words or tokens

removing punctuation
converting all words to lowercase.
-

Vocabulary Creation
A vocabulary of all unique words in the entire corpus (collection of documents) is created.

Frequency Counting
The frequency of each word in the vocabulary is counted.

Vector Representation
Each document is then represented as a vector where each element corresponds to the frequency of a word in the vocabulary.

Term Frequency (TF)simplest form of BoW uses the raw count of each word as its frequency. This is called Term Frequency.

2 ways building-BoW

1. **Creating a Vocabulary and Assigning Unique Integer Indices**
Gather all the unique words from the entire corpus (collection of documents) to create a vocabulary. This vocabulary forms the foundation of the BoW model.

Each word in the vocabulary is assigned a unique integer index. This index is used to represent the position of each word in the vector.
- **Example :** , if the vocabulary is ["apple", "banana", "cherry"], "apple" might have index 0, "banana" index 1, and "cherry" index 2.

2. Creating a Vector Recording the Frequency of Word Tokens at Each Index Position
Each document is represented as a vector. The size of the vector is equal to the size of the vocabulary.

Each element of the vector represents the frequency of the word in the vocabulary corresponding to that index in the document.

Example
if the vocabulary is ["apple", "banana", "cherry"] and a document contains the text "apple banana apple", the BoW vector for this document would be [2, 1, 0]. (This indicates that "apple" appears 2 times, "banana" appears 1 time, and "cherry" appears 0 times.)

How to build BoW

Imports and initialization

```
From konlpy.tag import out

okt= okt()
```

imports the Okt class from the konlpy.tag module
konlpy = python library for korean natural language processing
Okt = okt() -> creates an instances of 0kt classes, will be used to perform morphological analysis on the input text

Text preprocessing

```
# removes all periods(.)
document = document.replace('.', '')

# lowercase
document = document.lower()
```

Tokenization

```
tokenized_document = okt.morphs(document)
```

Initialization of Data Structure

```
word_to_index = {}
  bow = []
```

word_to_index = {}: This creates an empty dictionary called word_to_index
Bow = []: creates an empty list called bow., this list will stroe the BoW vector

Building the Bow

```
for word in tokenized_document:
    # remove stop words

    if stop_words is not None and word in stop_words:
        continue

    if word not in word_to_index.keys():
        word_to_index[word] = len(word_to_index)
        bow.insert(len(word_to_index) - 1, 1)
    else:
        index = word_to_index.get(word)
        bow[index] = bow[index] + 1
```

iterates through each word (token) in the tokenized_document list.

If stop_words is not None and word in stop_words:
Continue
Checks if a list of stop_words was provided and if the current word is in that list
If both conditions are true, the Continue : skips the next iteration of the loop

If word not in word_to_index.keys(): #checks if the current word is already in the word_to_index

If word not in word_to_index.keys(
word_to_index[word] = len(word_to_index)
bow.insert(len(word_to_index) - 1, 1)
If word not in the dic, run
word_to_index[word] = len(word_to_index) => The word is added to the word_to_index dictionary, and it's assigned a unique integer index

else: # the index of word is already in the dictionary
index = word_to_index.get(word)
bow[index] = bow[index] + 1

bow.insert(len(word_to_index) - 1, 1)

ensures that the new word's count is initialized to
1. **insert** is used to ensure that the elements are in the correct order.

Return values

```
return word_to_index, bow
```

retrieved from the word_to_index dictionary.
- bow[index] = bow[index] + 1: The count for that word at the corresponding index in the bow list is incremented by 1.