Vocabulary

Become accustomed to the meaning of "vocabulary" for NLP tasks.

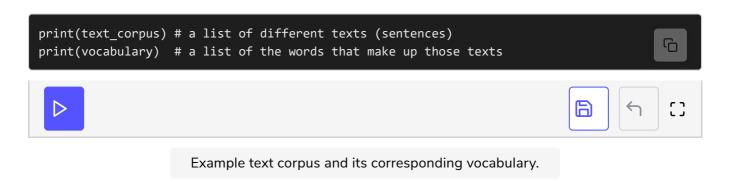
Chapter Goals:

- Learn about the text corpus and vocabulary in NLP tasks
- Create a function that tokenizes a text corpus

A. Corpus vocabulary

In the context of NLP tasks, the **text corpus** refers to the set of texts used for the task. For example, if we were building a model to analyze news articles, our text corpus would be the entire set of articles or papers we used to train and evaluate the model.

The set of unique words used in the text corpus is referred to as the **vocabulary**. When processing raw text for NLP, everything is done around the vocabulary.



In addition to using the words of a text corpus as the vocabulary, you could also use a **character-based** vocabulary. This would consist of each unique character in the text corpus (e.g. each letter). In this course, we'll be focusing on **word-based vocabularies**, which are much more common than their character-based counterparts.

B. Tokenization

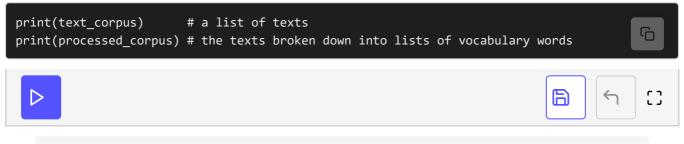
We can use the vocabulary to find the number of times each word appears in

filter each text document based on the words that appear in it. However, the

most important part of the vocabulary is that it allows us to represent each piece of text by the specific words that appear in it.

Rather than being represented as one long string, a piece of text can be represented as a vector/list of its vocabulary words. This process is known as **tokenization**, where each individual vocabulary word in a piece of text is a **token**.

Below we show an example of tokenization on a text corpus.



Representing pieces of text by vocabulary words. Note that the punctuation is filtered out.

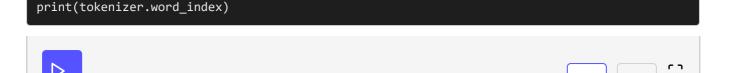
In the example above, the punctuation is filtered out of the text corpus. While it is normally standard to filter out punctuation, in some cases (e.g. generating long text) it may be necessary to keep punctuation in the vocabulary. It is a good idea to understand the NLP task you are going to perform before filtering out any data/piece of text.

C. Tokenizer object

Using TensorFlow, we can convert a text corpus into tokenized sequences using the Tokenizer object. The Tokenizer class is part of the tf.keras submodule, which is TensorFlow's implementation of Keras, a high-level API for machine learning.

The Tokenizer object contains the functions fit_on_texts and texts_to_sequences, which are used to initialize the object with a text corpus and convert pieces of text into sequences of tokens, respectively.

```
import tensorflow as tf
tokenizer = tf.keras.preprocessing.text.Tokenizer()
text_corpus = ['bob ate apples, and pears', 'fred ate apples!']
tokenizer.fit_on_texts(text_corpus)
new_texts = ['bob ate pears', 'fred ate pears']
print(tokenizer.texts_to_sequences(new_texts))
```



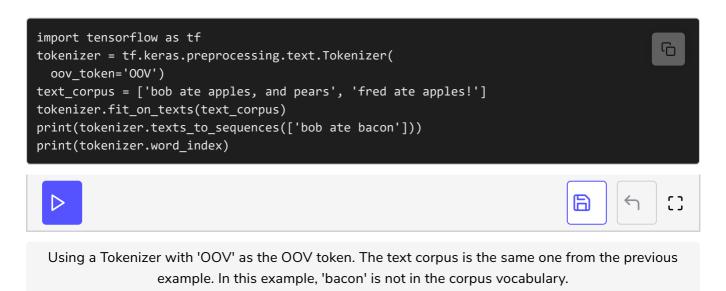
Using a Tokenizer object for text processing. Notice that it maps vocabulary words to indexes.

The Tokenizer automatically converts each vocabulary word to an integer ID (IDs are given to words by descending frequency). This allows the tokenized sequences to be used in NLP algorithms (which work on vectors of numbers). In the above example, the texts_to_sequences function converts each vocabulary word in new_texts to its corresponding integer ID.

D. Tokenizer parameters

The Tokenizer object can be initialized with a number of optional parameters. By default, the Tokenizer filters out any punctuation and white space. You can specify custom filtering with the filters parameter. The parameter takes in a string, where each character in the string is filtered out.

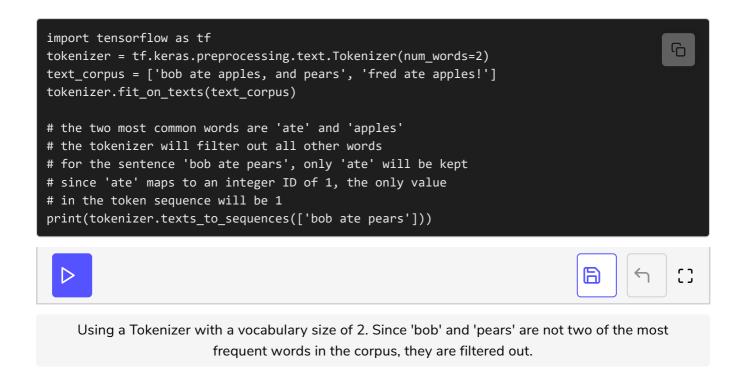
When a new text contains words not in the corpus vocabulary, those words are known as out-of-vocabulary (OOV) words. The texts_to_sequences automatically filters out all OOV words. However, if we want to specify each OOV word with a special vocabulary token (e.g. 'OOV'), we can initialize the Tokenizer with the oov_token parameter.



The num_words parameter lets us specify the maximum number of vocabulary words to use. For example, if we set num_words=100 when initializing the Tokenizer, it will only use the 100 most frequent words in the vocabulary and filter out the remaining vocabulary words. This can be useful when the text

med out the femalism of vocasitary words. This can be account when the tent

corpus is large and you need to limit the vocabulary size to increase training speed or prevent overfitting on infrequent words.



Time to Code!

The code for this section of the course involves building up an embedding model. Specifically, you will be building out the EmbeddingModel object. In this chapter, you'll be completing the tokenize_text_corpus function.

You'll notice that in the model initialization, the Tokenizer object is already set, with its maximum vocabulary size set to vocab_size. However, the Tokenizer object has not yet been initialized with a text corpus.

In the tokenize_text_corpus function, we'll first initialize the Tokenizer with the text corpus, texts.

```
Call self.tokenizer.fit_on_texts on texts.
```

After initializing the Tokenizer with the text corpus, we can use it to convert the text corpus into tokenized sequences.

Set sequences equal to self.tokenizer.texts_to_sequences applied to texts.

Then return sequences.

```
# Skip-gram embedding model
class EmbeddingModel(object):
    # Model Initialization

def __init__(self, vocab_size, embedding_dim):
    self.vocab_size = vocab_size
    self.embedding_dim = embedding_dim
    self.tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=self.vocab_size)

# Convert a list of text strings into word sequences
def tokenize_text_corpus(self, texts):
    # CODE HERE
    pass
```









[]