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.

```
print(text_corpus) # a list of different texts (sentences)
print(vocabulary)  # a list of the words that make up those texts
```

RUN SAVE RESET

Close

Output 0.486s

```
['bob ate apples', 'fred ate apples', 'bob ate pears']
['bob', 'fred', 'ate', 'apples', 'pears']
```

Example text corpus and its corresponding 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 the corpus, figure out which words are the most common or uncommon, and 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.

```
print(text_corpus)      # a list of texts
print(processed_corpus) # the texts broken down into lists of vocabulary words
```

RUN SAVE RESET

Close

Output 0.455s

```
['bob ate apples, and pears', 'fred ate apples!']
[['bob', 'ate', 'apples', 'and', 'pears'], ['fred', 'ate', 'apples']]
```

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))
print(tokenizer.word_index)
```

RUN SAVE RESET

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.

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

RUN SAVE RESET

Close

Output 5.138s

```
[[4, 2, 1]]
{'OOV': 1, 'ate': 2, 'apples': 3, 'bob': 4, 'and': 5, 'pears': 6, 'fred': 7}
```

Using a Tokenizer with 'OOV' as the OOV token. The text corpus is the same one from the previous example. In this example, 'bacon' is not in the corpus vocabulary.

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

```python
import tensorflow as tf
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=2)
text_corpus = ['bob ate apples, and pears', 'fred ate apples!']
tokenizer.fit_on_texts(text_corpus)

# the two most common words are 'ate' and 'apples'
# the tokenizer will filter out all other words
# for the sentence 'bob ate pears', only 'ate' will be kept
# since 'ate' maps to an integer ID of 1, the only value
# in the token sequence will be 1
print(tokenizer.texts_to_sequences(['bob ate pears']))
```

Output 4.907s

```
[[1]]
```

Using a Tokenizer with a vocabulary size of 2. Since 'bob' and 'pears' are not two of the most frequent words in the corpus, they are filtered out.