**Introduction and installation of packages**

**Natural Language Processing** (**NLP**) has become an important part of modern systems. It is used extensively in search engines, conversational interfaces, document processors, and so on. Machines can handle structured data well, but when it comes to working with free-form text, they have a hard time. The goal of NLP is to develop algorithms that enable computers to understand free - form text and help them understand language.

One of the most challenging things about processing free - form natural language is the sheer amount of variation. Context plays a very important role in how a sentence is understood. Humans are innately great at understanding language. It is not clear yet how humans understand language so easily and intuitively. We use our past knowledge and experiences to understand conversations and we can quickly get the gist of what other people are talking about even with little explicit context.

To address this issue, NLP researchers started developing various applications using machine learning approaches. To build such applications, a large corpus of text is obtained and then algorithms are trained on this data to perform various tasks such as categorizing text, analyzing sentiments, and modeling topics. The algorithms are trained to detect patterns in the input text data and to derive insights from it.

In this chapter, we will discuss various underlying concepts that are used to analyze text and build NLP applications. This will enable us to understand how to extract meaningful information from the given text data. We will use a Python package called **Natural Language Toolkit** (**NLTK**) to build these applications. You can install it by running the following command:

**$ pip install nltk**

You can find more information about NLTK at [http://www.nltk.org](http://www.nltk.org/).

In order to access all the datasets provided by NLTK, we need to download it. Open a Python shell by typing the following:

**$ python**

We are now inside the Python shell. Type the following to download the data:

**>>> import nltk**

**>>> nltk.download()**

We will also use a package called gensim in this chapter. gensim is a robust semantic modeling library that's useful for many applications. It can be installed by running the following command:

**$ pip install gensim**

You might need another package, called pattern, for gensim to function properly. You can install it by running the following command:

**$ pip3 install pattern**

You can find more information about gensim at <https://radimrehurek.com/gensim>. Now that you have installed the NLTK and gensim, let's proceed with the discussion.

**Tokenizing text data**

When we deal with text, we need to break it down into smaller pieces for analysis. To do this, tokenization can be applied. Tokenization is the process of dividing text into a set of pieces, such as words or sentences. These pieces are called tokens. Depending on what we want to do, we can define our own methods to divide the text into many tokens. Let's look at how to tokenize the input text using NLTK.

**Create a new Python file and import the following packages:**

from nltk.tokenize import sent\_tokenize, \

word\_tokenize, WordPunctTokenizer

**Define the input text that will be used for tokenization:**

*# Define input text*

input\_text = "Do you know how tokenization works? It's actually \

quite interesting! Let's analyze a couple of sentences and \

figure it out."

**Divide the input text into sentence tokens:**

# Sentence tokenizer

print("\nSentence tokenizer:")

print(sent\_tokenize(input\_text))

**Divide the input text into word tokens:**

# Word tokenizer

print("\nWord tokenizer:")

print(word\_tokenize(input\_text))

**Divide the input text into word tokens using the WordPunct tokenizer:**

# WordPunct tokenizer

print("\nWord punct tokenizer:")

print(WordPunctTokenizer().tokenize(input\_text))

The full code is given in the file tokenizer.py. If you run the code, you will get the following output:

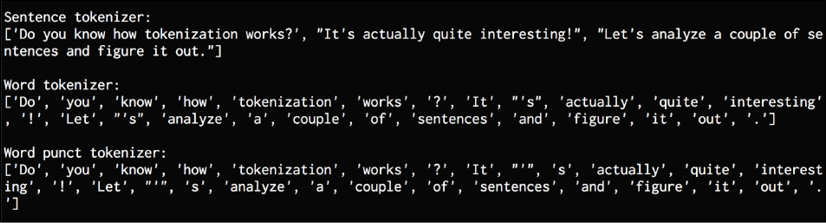


Figure 1: Tokenizers output

The sentence tokenizer divides the input text into sentences. Two-word tokenizers behave differently when it comes to punctuation. For example, the word "It's" is divided differently by the punct tokenizer than by the regular tokenizer.