Tokenization

Posted May 3, 2023 • Updated May 12, 2023

By Shivam Solanki

6 min read

Tokenization

The process of tokenization involves dividing a text or a string of characters into tokens. The most typical form of tokenization used in natural language processing is breaking down a sentence or document into individual words, but it can also involve dividing text into other units like subwords, phrases, or even characters. Tokenization is frequently an essential step in language processing jobs where the individual units of text are evaluated and processed further, such as machine translation, sentiment analysis, and text categorization.

Types of Tokenizers

Word based tokenization

The technique of breaking down a text or a string of characters into discrete words or word-like units, or tokens, is known as word-based tokenization. In this kind of tokenization, each word or group of words forms a separate token and the document or sentence is typically segmented based on whitespace and punctuation. For example, the sentence “The quick brown fox jumped over the lazy dog” can be tokenized into individual words as follows: “The”, “quick”, “brown”, “fox”, “jumped”, “over”, “the”, “lazy”, “dog”.

One of the most often used tokenization techniques in natural language processing is word-based tokenization, which makes it simple to analyze text data at the word level. However, it might not always be suitable for scripts or languages that don’t use spaces to separate words, or for texts that contain special characters, symbols, or numerical values that should be treated as separate tokens. In such cases, alternative forms of tokenization, such as character or subword-based tokenization, may be more appropriate.

Split on spaces

Let’s	do	tokenization!

Split on punctuation

Let	’s	do	tokenization	!

Character based tokenization

Character-based tokenization is the process of breaking down a text or a sequence of characters into individual characters, rather than into words or other higher-level units. In this type of tokenization, each character in a document or a sentence is treated as a separate token, regardless of whether it is a letter, a number, a punctuation mark, or a special symbol. For example, the sentence “The quick brown fox jumped over the lazy dog” can be tokenized into individual characters as follows: “T”, “h”, “e”, “ “, “q”, “u”, “i”, “c”, “k”, “ “, “b”, “r”, “o”, “w”, “n”, “ “, “f”, “o”, “x”, “ “, “j”, “u”, “m”, “p”, “e”, “d”, “ “, “o”, “v”, “e”, “r”, “ “, “t”, “h”, “e”, “ “, “l”, “a”, “z”, “y”, “ “, “d”, “o”, “g”.

Character-based tokenization can be useful in some cases where word-based tokenization may not be appropriate or effective, such as for languages or scripts that do not use spaces to separate words, or for texts that contain special characters, symbols, or numerical values that should be treated as separate tokens. It can also be used as a pre-processing step in certain types of natural language processing tasks, such as text classification or sentiment analysis, to analyze text data at the character level.

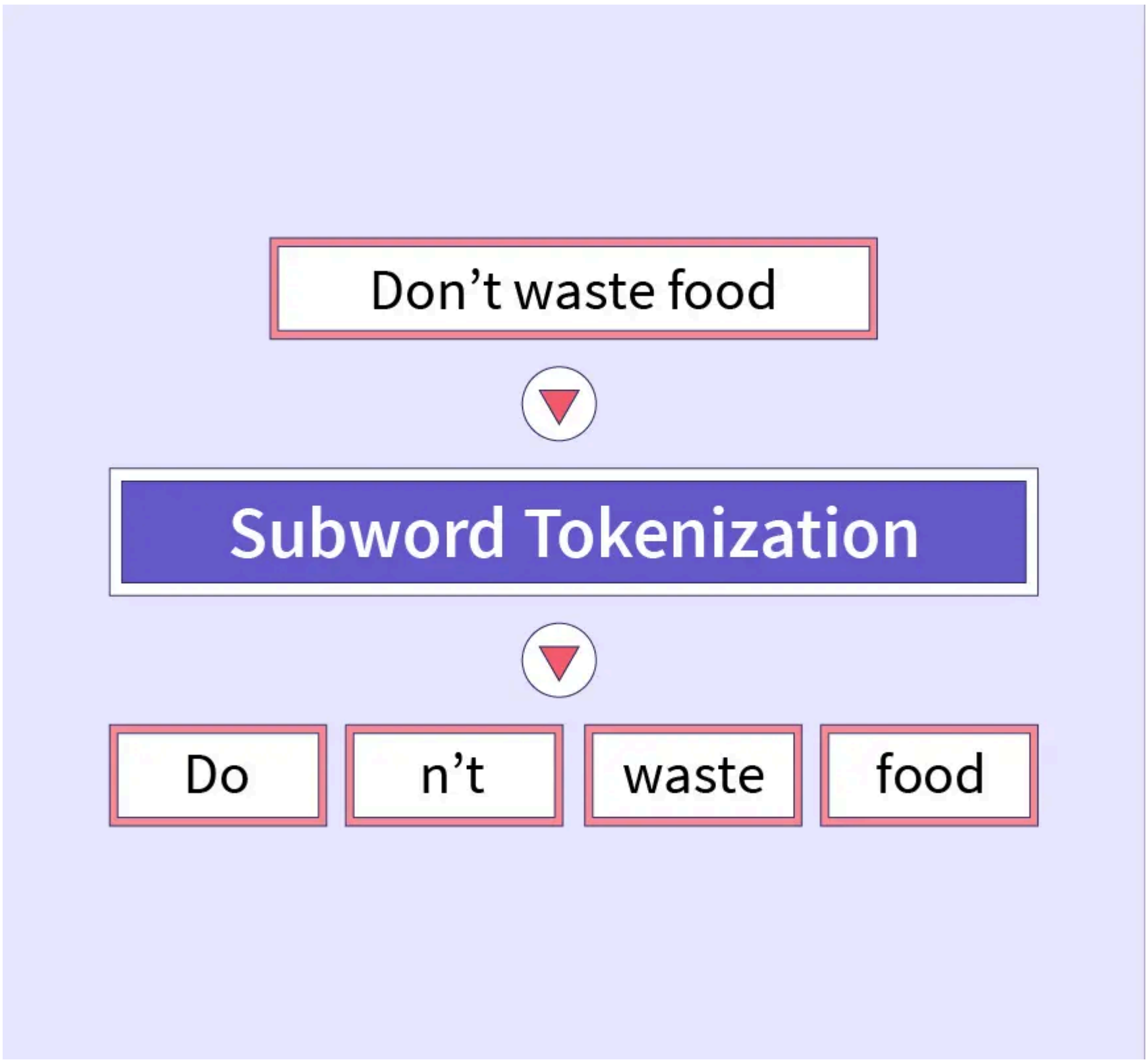
L e t ‘ s d o t o k e n i z a t i o n !

Subword tokenization

Subword tokenization is a type of tokenization that breaks down a text or a sequence of characters into smaller units called subwords, rather than into words or individual characters. In subword tokenization, the text is segmented into units that appear frequently in the data, and that are not necessarily whole words. These subwords may be partial words, or they may correspond to prefixes or suffixes that occur frequently in the data.

Subword tokenization is particularly useful for handling languages or scripts that have complex morphology, where words may be composed of multiple morphemes or subword units. For example, in German, the word “unwahrscheinlich” (meaning “improbable”) can be broken down into subwords such as “un-“, “wahr-“, “schein-“, and “-lich”, which can be treated as separate units for analysis or processing. In English, subword tokenization can be used to handle words with common prefixes or suffixes, such as “un-“, “re-“, “-ing”, and “-ed”.

Subword tokenization is commonly used in modern natural language processing models, particularly in the context of neural machine translation and language modeling, where it has been shown to improve performance on tasks involving rare or out-of-vocabulary words. It is often implemented using algorithms such as byte-pair encoding (BPE) or sentencepiece, which learn the most frequent subword units from the data through an iterative process.



In natural language processing, there are different types of tokenizers that can be used for various text processing tasks. Some of the commonly used tokenizer types include:

Rule-based tokenizers: These tokenizers use a set of pre-defined rules to split the text into tokens. For example, a rule-based tokenizer may split the text at every whitespace character, punctuation mark, or special character.

Statistical tokenizers: These tokenizers use statistical models to learn how to split the text into tokens. For example, a statistical tokenizer may learn from a large corpus of text data to identify frequently occurring word boundaries and use them to split the text into tokens.

Neural network-based tokenizers: These tokenizers use neural networks to learn how to split the text into tokens. For example, a neural network-based tokenizer may use a recurrent neural network to predict the likelihood of a particular character or sequence of characters being a token boundary.

Hybrid tokenizers: These tokenizers combine multiple approaches, such as rule-based and statistical or neural network-based, to achieve more accurate and robust tokenization results.

The choice of tokenizer type depends on the specific requirements of the text processing task and the characteristics of the text data. Rule-based tokenizers can be simple and fast, but may not work well with complex or diverse text data. Statistical and neural network-based tokenizers can be more accurate and adaptable, but may require more computational resources and training data. Hybrid tokenizers can offer a balance between accuracy and efficiency by leveraging the strengths of different approaches.

Using Tokenizers

Loading tokenizer

Load the tokenizer from `BertTokenizer` class

</> Plaintext

```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained("bert-base-cased")
```

or load the tokenizer from the `AutoTokenizer` class

</> Plaintext

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
```

An example output is shown below:

</> Plaintext

```
{'input_ids': [101, 7993, 170, 11303, 1200, 2443, 1110, 3014, 102],
 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0],
 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

Save the tokenizer

</> Plaintext

```
tokenizer.save_pretrained("directory_on_my_computer")
```

Plaintext

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

sequence = "Using a Transformer network is simple"
tokens = tokenizer.tokenize(sequence)

print(tokens)
```

The output of the above subword tokenizer is shown below

Plaintext

```
['Using', 'a', 'transform', '##er', 'network', 'is', 'simple']
```

Now you can use the token to convert them to input IDs using

Plaintext

```
ids = tokenizer.convert_tokens_to_ids(tokens)

print(ids)
```

These outputs can then be used in the models.

Decoding using tokenizer

Decoding is the reverse of encoding.

Plaintext

```
decoded_string = tokenizer.decode([7993, 170, 11303, 1200, 2443, 1110, 3014])
print(decoded_string)
```

The `decode` method converts indices to tokens and also group them together to produce a meaningful sentence.

Reference

Hugging Face

This post is licensed under CC BY 4.0 by the author.

Share:

Further Reading

May 10, 2023

May 3, 2023

May 2, 2023

Re-ranker ColBERT Re-ranker ColBERT is a technique that creates separate detailed…

Getting started with transformer models The AutoModel class is a tool used to create a…

Exploring Techniques for Tuning Large Language Models (LLMs) As the field of…

OLDER
Getting Started With Transformer Models

NEWER
Re Ranker

© 2023 Shivam Solanki. Some rights reserved.

Using the Jekyll theme Chirpy.