Package 'tok'

August 18, 2023

Title Fast Text Tokenization

Version 0.1.1

Description Interfaces with the 'Hugging Face' tokenizers library to provide implementations of today's most used tokenizers such as the 'Byte-Pair Encoding' algorithm <https://huggingface.co/docs/tokenizers/index>. It's extremely fast for both training new vocabularies and tokenizing texts.

License MIT + file LICENSE

SystemRequirements Rust tool chain w/ cargo, libclang/llvm-config

Encoding UTF-8

RoxygenNote 7.2.3

Depends R (>= 4.2.0)

Imports R6, cli

Suggests rmarkdown, testthat (>= 3.0.0), hfhub, withr

Config/testthat/edition 3

URL https://github.com/mlverse/tok

BugReports https://github.com/mlverse/tok/issues

NeedsCompilation yes

Author Daniel Falbel [aut, cre],
Posit [cph]

Maintainer Daniel Falbel <daniel@posit.co>

Repository CRAN

Date/Publication 2023-08-17 23:30:02 UTC

R topics documented:

encoding 2
tokenizer 3

Index **6**

encoding *Encoding*

Description

Represents the output of a tokenizer.

Value

An encoding object containing encoding information such as attention masks and token ids.

Public fields

`.encoding` The underlying implementation pointer.

Active bindings

`ids` The IDs are the main input to a Language Model. They are the token indices, the numerical representations that a LM understands.

`attention_mask` The attention mask used as input for transformers models.

Methods

Public methods:

- `encoding$new()`
- `encoding$clone()`

Method `new()`**:** Initializes an encoding object (Not to use directly)

Usage:

```
encoding$new(encoding)
```

Arguments:

`encoding` an encoding implementation object

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
encoding$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
withr::with_envvar(c(HUGGINGFACE_HUB_CACHE = tempdir()), {
try({
tok <- tokenizer$from_pretrained("gpt2")
encoding <- tok$encode("Hello world")
encoding
})
})
```

`tokenizer` *Tokenizer*

Description

A Tokenizer works as a pipeline. It processes some raw text as input and outputs an encoding.

Value

A tokenizer that can be used for encoding character strings or decoding integers.

Public fields

`.tokenizer` (unsafe usage) Lower level pointer to tokenizer

Methods

Public methods:

- `tokenizer$new()`
- `tokenizer$encode()`
- `tokenizer$decode()`
- `tokenizer$encode_batch()`
- `tokenizer$decode_batch()`
- `tokenizer$from_file()`
- `tokenizer$from_pretrained()`
- `tokenizer$clone()`

Method `new()`**:** Initializes a tokenizer

Usage:

```
tokenizer$new(tokenizer)
```

Arguments:

`tokenizer` Will be cloned to initialize a new tokenizer

Method `encode()`**:** Encode the given sequence and pair. This method can process raw text sequences as well as already pre-tokenized sequences.

Usage:

```
tokenizer$encode(
  sequence,
  pair = NULL,
  is_pretokenized = FALSE,
  add_special_tokens = TRUE
)
```

Arguments:

`sequence` The main input sequence we want to encode. This sequence can be either raw text or pre-tokenized, according to the is_pretokenized argument

`pair` An optional input sequence. The expected format is the same that for sequence.

`is_pretokenized` Whether the input is already pre-tokenized

`add_special_tokens` Whether to add the special tokens

Method `decode()`**:** Decode the given list of ids back to a string

Usage:

```
tokenizer$decode(ids, skip_special_tokens = TRUE)
```

Arguments:

`ids` The list of ids that we want to decode

`skip_special_tokens` Whether the special tokens should be removed from the decoded string

Method `encode_batch()`**:** Encodes a batch of sequences. Returns a list of encodings.

Usage:

```
tokenizer$encode_batch(
  input,
  is_pretokenized = FALSE,
  add_special_tokens = TRUE
)
```

Arguments:

`input` A list of single sequences or pair sequences to encode. Each sequence can be either raw text or pre-tokenized, according to the is_pretokenized argument.

`is_pretokenized` Whether the input is already pre-tokenized

`add_special_tokens` Whether to add the special tokens

Method `decode_batch()`**:** Decode a batch of ids back to their corresponding string

Usage:

```
tokenizer$decode_batch(sequences, skip_special_tokens = TRUE)
```

Arguments:

`sequences` The batch of sequences we want to decode

`skip_special_tokens` Whether the special tokens should be removed from the decoded strings

Method `from_file()`**:** Creates a tokenizer from the path of a serialized tokenizer. This is a static method and should be called instead of `$new` when initializing the tokenizer.

Usage:

```
tokenizer$from_file(path)
```

Arguments:

`path` Path to tokenizer.json file

Method `from_pretrained()`**:** Instantiate a new Tokenizer from an existing file on the Hugging Face Hub.

Usage:

```
tokenizer$from_pretrained(identifier, revision = "main", auth_token = NULL)
```

Arguments:

`identifier` The identifier of a Model on the Hugging Face Hub, that contains a tokenizer.json file

`revision` A branch or commit id

`auth_token` An optional auth token used to access private repositories on the Hugging Face Hub

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
tokenizer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
withr::with_envvar(c(HUGGINGFACE_HUB_CACHE = tempdir()), {
try({
tok <- tokenizer$from_pretrained("gpt2")
tok$encode("Hello world")$ids
})
})
```

Index

encoding, 2, *3*, *4*

tokenizer, *2*, 3