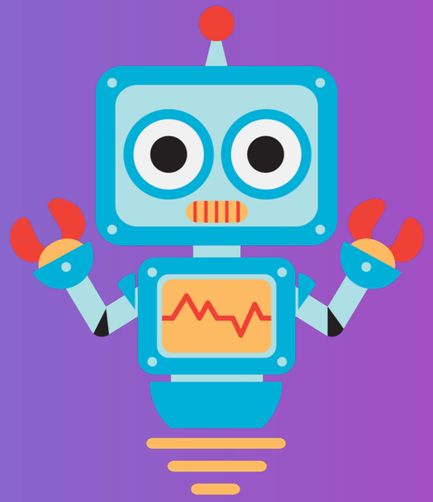
Dr. Abulkarim Albanna

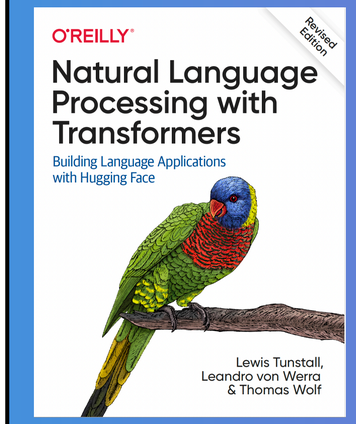
Text Classification

https://huggingface.co/docs/transformers/model_doc/distilbert



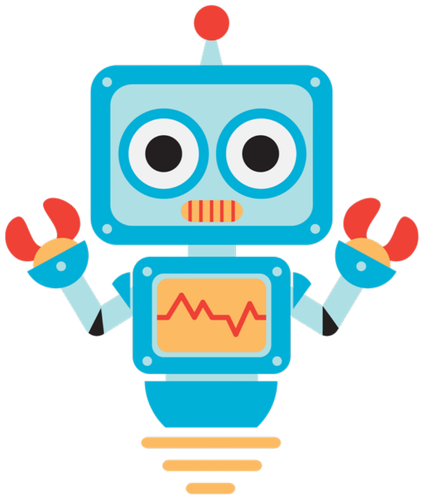
Day 3

January 18th 2024



Content

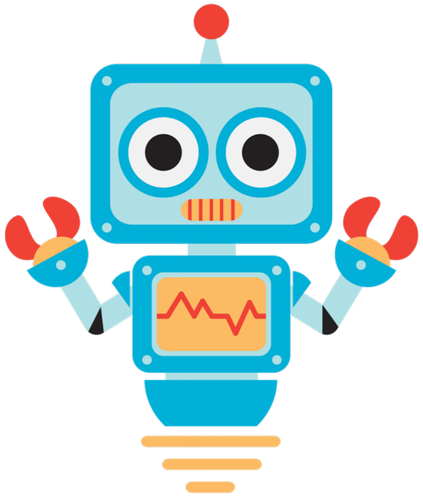
- Introduction
- A First Look at Hugging Face Datasets
- From Datasets to DataFrames
- Looking at the Class Distribution
- From Text to Tokens
- Feature extraction
- Fine-Tuning Transformers



Introduction

Text classification

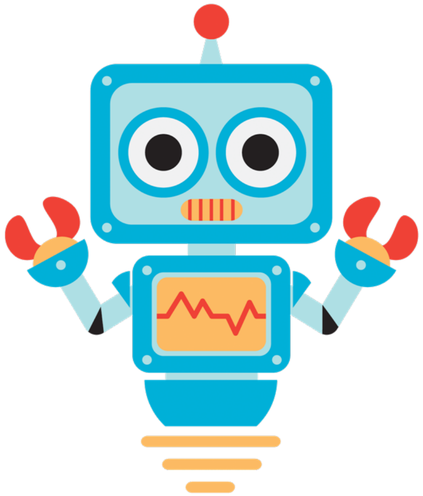
Text classification is one of the most common tasks in NLP; it can be used for a broad range of applications, such as **tagging customer feedback** into categories or routing support tickets according to their language. Chances are that your email program's **spam filter** is using text classification to protect your inbox from a junk!. Another common type of text classification is **sentiment analysis.**



Introduction

Text classification

As a data scientist who needs to build a system that can automatically identify **emotional** states such as “anger” or “joy” that people express about your company’s product on Twitter.

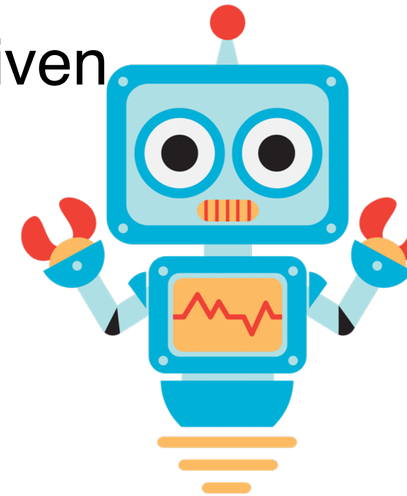


Introduction

Text classification

- We'll tackle this task using a variant of BERT called **DistilBERT**.The main advantage of this model is that it achieves comparable performance to BERT, while being significantly smaller and more efficient. This enables us to train a classifier in a few minutes, and if you want to train a larger **BERT** model you can simply change the **checkpoint** of the pretrained model.

- A checkpoint corresponds to the set of weights that are loaded into a given transformer architecture.

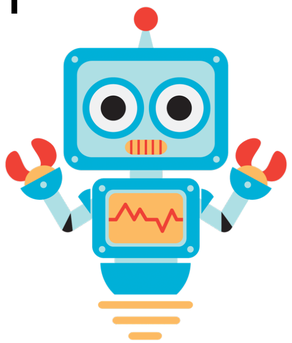


Introduction

Text classification

- We'll tackle this task using a variant of BERT called **DistilBERT**.The main advantage of this model is that it achieves comparable performance to BERT, while being significantly smaller and more efficient. This enables us to train a classifier in a few minutes, and if you want to train a larger **BERT** model you can simply change the **checkpoint** of the pretrained model.

- A checkpoint corresponds to the set of weights that are loaded into a given transformer
architecture.



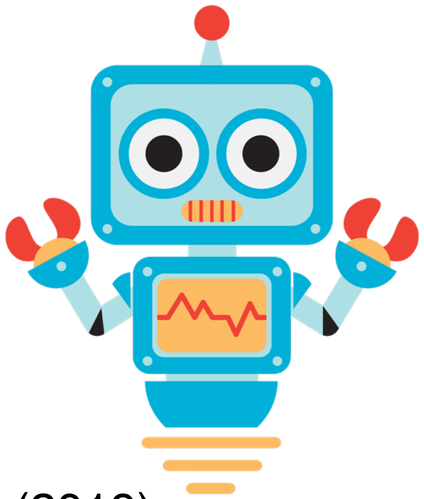
V. Sanh et al., "DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter", (2019).

Introduction

Text classification

Table 1: **DistilBERT retains 97% of BERT performance.** Comparison on the dev sets of the GLUE benchmark. ELMo results as reported by the authors. BERT and DistilBERT results are the medians of 5 runs with different seeds.

Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

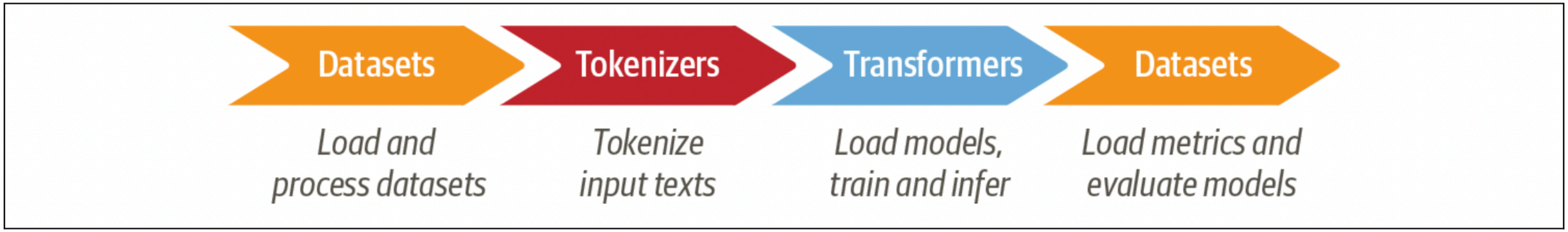


https://github.com/huggingface/transformers

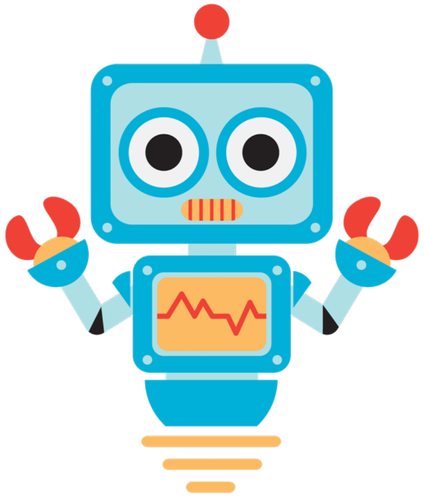
V. Sanh et al., “DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter”, (2019).

Introduction

Text classification



A typical pipeline for training transformer models with the Datasets, Tokenizers, and Transformers libraries



Text classification

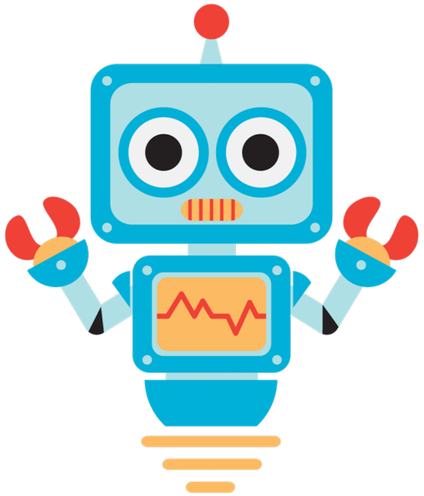
A First Look at Hugging Face Datasets

We will use Datasets to download the data from the Hugging Face Hub. We can use the **`list_datasets()`** function to see what datasets are available on the Hub:

```
from datasets import list_datasets

all_datasets = list_datasets()
print(f"There are {len(all_datasets)} datasets currently available on the Hub")
print(f"The first 10 are: {all_datasets[:10]}")
```

```
There are 1753 datasets currently available on the Hub
The first 10 are: ['acronym_identification', 'ade_corpus_v2', 'adversarial_qa',
'aeslc', 'afrikaans_ner_corpus', 'ag_news', 'ai2_arc', 'air_dialogue',
'ajgt_twitter_ar', 'allegro_reviews']
```



Text classification

A First Look at Hugging Face Datasets

We see that each dataset is given a name, so let's load the `emotion` dataset with the **`load_dataset()`** function:

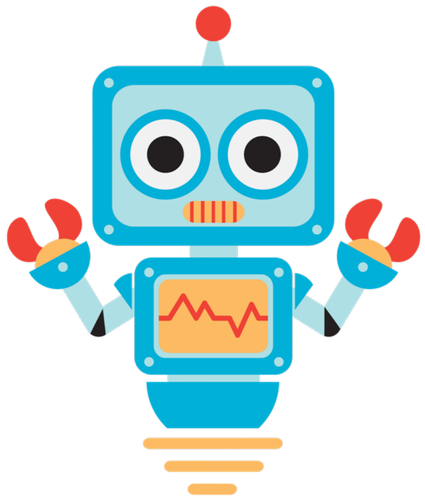
```
from datasets import load_dataset

emotions = load_dataset("emotion")
```

If we look inside our `emotions` object:

```
emotions
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 16000
    })
    validation: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
})
```



Text classification

A First Look at Hugging Face Datasets

We will use Datasets to download the data from the Hugging Face Hub. We can use the **`list_datasets()`** function to see what datasets are available on the Hub:

Table 2-1. How to load datasets in various formats

Data format	Loading script	Example
CSV	`csv`	`load_dataset("csv", data_files="my_file.csv")`
Text	`text`	`load_dataset("text", data_files="my_file.txt")`
JSON	`json`	`load_dataset("json", data_files="my_file.jsonl")`

Text classification

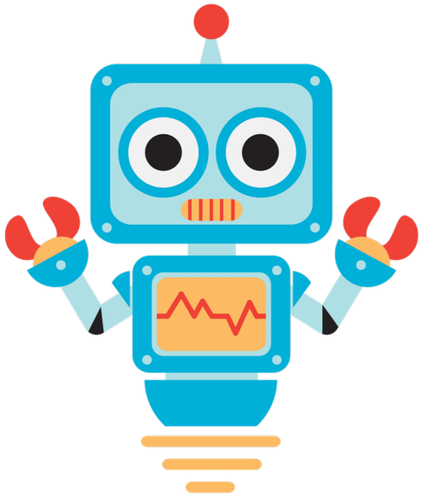
From Datasets to DataFrames

Convert a `Dataset` object to a Pandas `DataFrame` so we can access high-level APIs for data visualization

```
import pandas as pd

emotions.set_format(type="pandas")
df = emotions["train"][:]
df.head()
```

	text	label
0	i didnt feel humiliated	0
1	i can go from feeling so hopeless to so damned...	0
2	im grabbing a minute to post i feel greedy wrong	3
3	i am ever feeling nostalgic about the fireplac...	2
4	i am feeling grouchy	3



Text classification

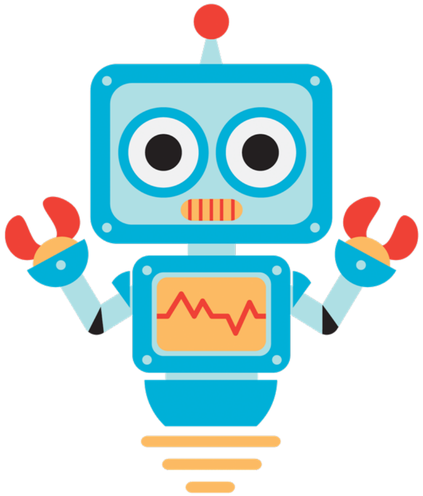
From Datasets to DataFrames

Convert a `Dataset` object to a Pandas `DataFrame` so we can access high-level APIs for data visualization

```
def label_int2str(row):
    return emotions["train"].features["label"].int2str(row)

df["label_name"] = df["label"].apply(label_int2str)
df.head()
```

	text	label	label_name
0	i didnt feel humiliated	0	sadness
1	i can go from feeling so hopeless to so damned...	0	sadness



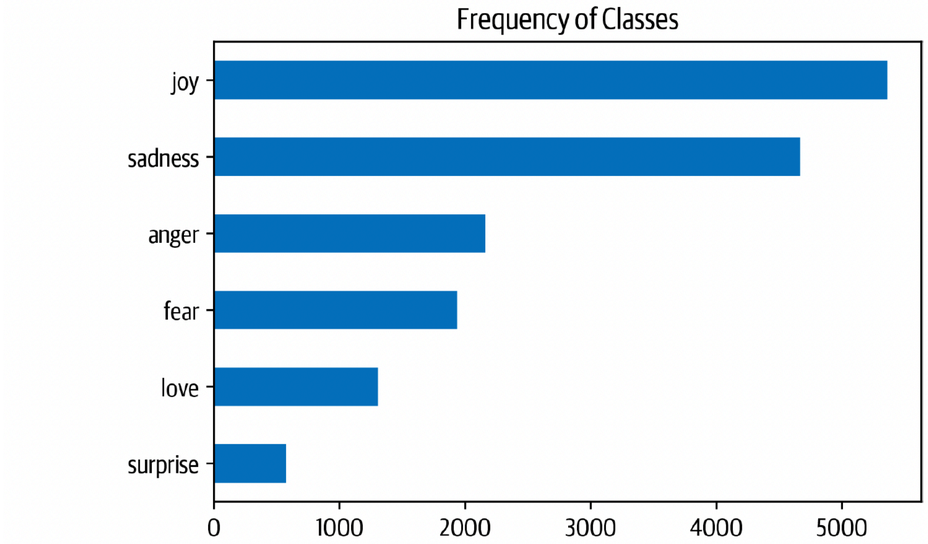
Text classification

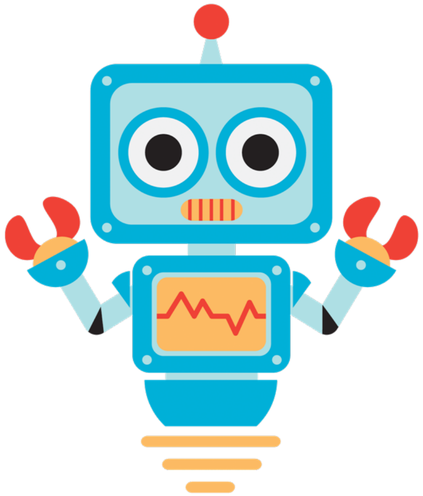
Looking at the Class Distribution

Whenever you are working on text classification problems, it is a good idea to examine the distribution of examples across the classes. A dataset with **imbalance distribution** might require a different treatment in terms of the training loss and evaluation metrics than a balanced one.

```
import matplotlib.pyplot as plt

df["label_name"].value_counts(ascending=True).plot.barh()
plt.title("Frequency of Classes")
plt.show()
```





Text classification

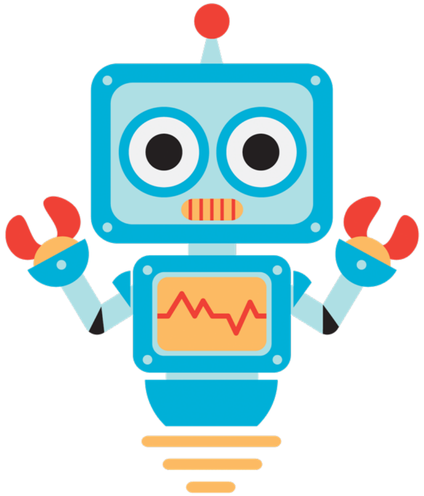
From Text to Tokens

Character Tokenization

The simplest tokenization scheme is to feed each character individually to the model. In Python, `str` objects are really arrays under the hood, which allows us to quickly implement character-level tokenization with just one line of code:

```
text = "Tokenizing text is a core task of NLP."
tokenized_text = list(text)
print(tokenized_text)
```

```
['T', 'o', 'k', 'e', 'n', 'i', 'z', 'i', 'n', 'g', ' ', 't', 'e', 'x', 't', ' ',
'i', 's', ' ', 'a', ' ', 'c', 'o', 'r', 'e', ' ', 't', 'a', 's', 'k', ' ', 'o',
'f', ' ', 'N', 'L', 'P', '.']
```



Text classification

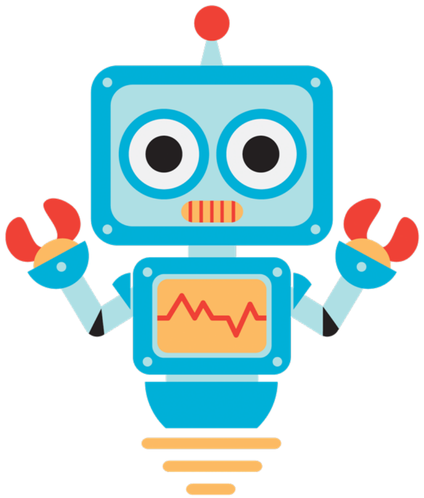
From Text to Tokens

Character Tokenization

The simplest tokenization scheme is to feed each character individually to the model. In Python, `str` objects are really arrays under the hood, which allows us to quickly implement character-level tokenization with just one line of code:

```
token2idx = {ch: idx for idx, ch in enumerate(sorted(set(tokenized_text)))}
print(token2idx)
```

```
{' ': 0, '.': 1, 'L': 2, 'N': 3, 'P': 4, 'T': 5, 'a': 6, 'c': 7, 'e': 8, 'f': 9,
'g': 10, 'i': 11, 'k': 12, 'n': 13, 'o': 14, 'r': 15, 's': 16, 't': 17, 'x': 18,
'z': 19}
```



Text classification

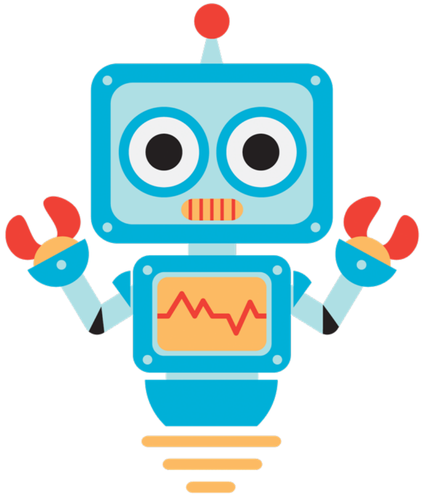
From Text to Tokens

Word Tokenization

Instead of splitting the text into characters, we can split it into words and map each word to an integer.

```
tokenized_text = text.split()
print(tokenized_text)
```

```
['Tokenizing', 'text', 'is', 'a', 'core', 'task', 'of', 'NLP.']
```

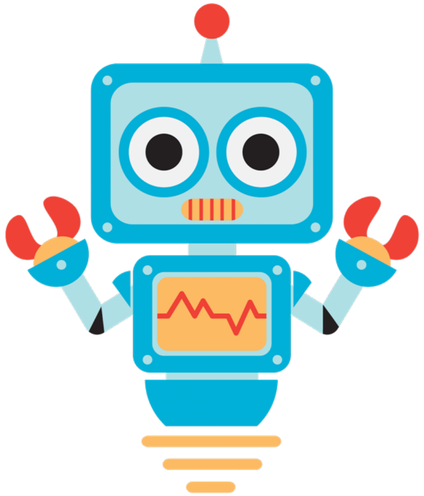


Text classification

From Text to Tokens

Subword Tokenization

There are several subword tokenization algorithms that are commonly used in NLP, but let's start with **WordPiece**,5 which is used by the **BERT** and **DistilBERT** tokenizers. Transformers provides a convenient `AutoTokenizer` class that allows you to quickly load the tokenizer associated with a pretrained model—we just call its **`from_pretrained()`** method, providing the ID of a model on the Hub or a local file path. Let's start by loading the tokenizer for DistilBERT:



Text classification

From Text to Tokens

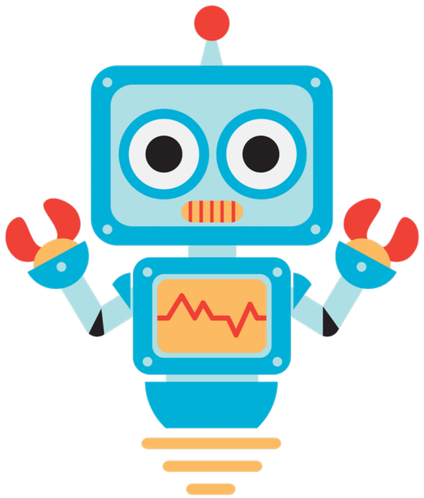
Subword Tokenization

```
from transformers import AutoTokenizer

model_ckpt = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
```

```
from transformers import DistilBertTokenizer

distilbert_tokenizer = DistilBertTokenizer.from_pretrained(model_ckpt)
```



Text classification

From Text to Tokens

Subword Tokenization

The `AutoTokenizer` class also has several attributes that provide information about the tokenizer. For example, we can inspect the vocabulary size:

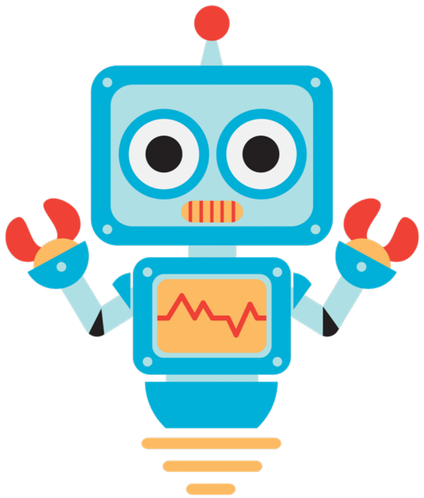
```
tokenizer.vocab_size
```

```
30522
```

and the corresponding model's maximum context size:

```
tokenizer.model_max_length
```

```
512
```



Text classification

From Text to Tokens

Tokenizing the Whole Dataset

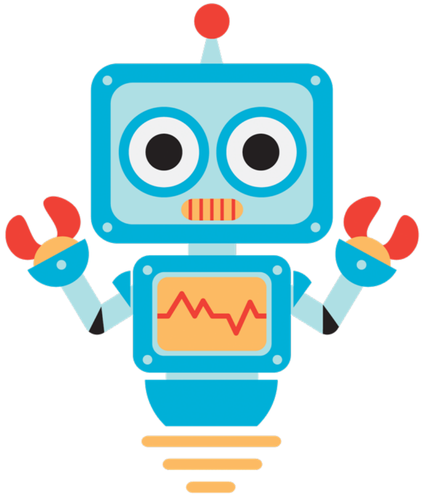
To tokenize the whole corpus, we'll use the **`map()`** method of our `DatasetDict` object. It provides a convenient way to apply a processing function to each element in a dataset.

```
def tokenize(batch):
    return tokenizer(batch["text"], padding=True, truncation=True)
```

This function applies the tokenizer to a batch of examples; **padding=True** will pad the examples with zeros

to the size of the longest one in a batch, and **truncation=True** will truncate the examples to the model's maximum

context size.

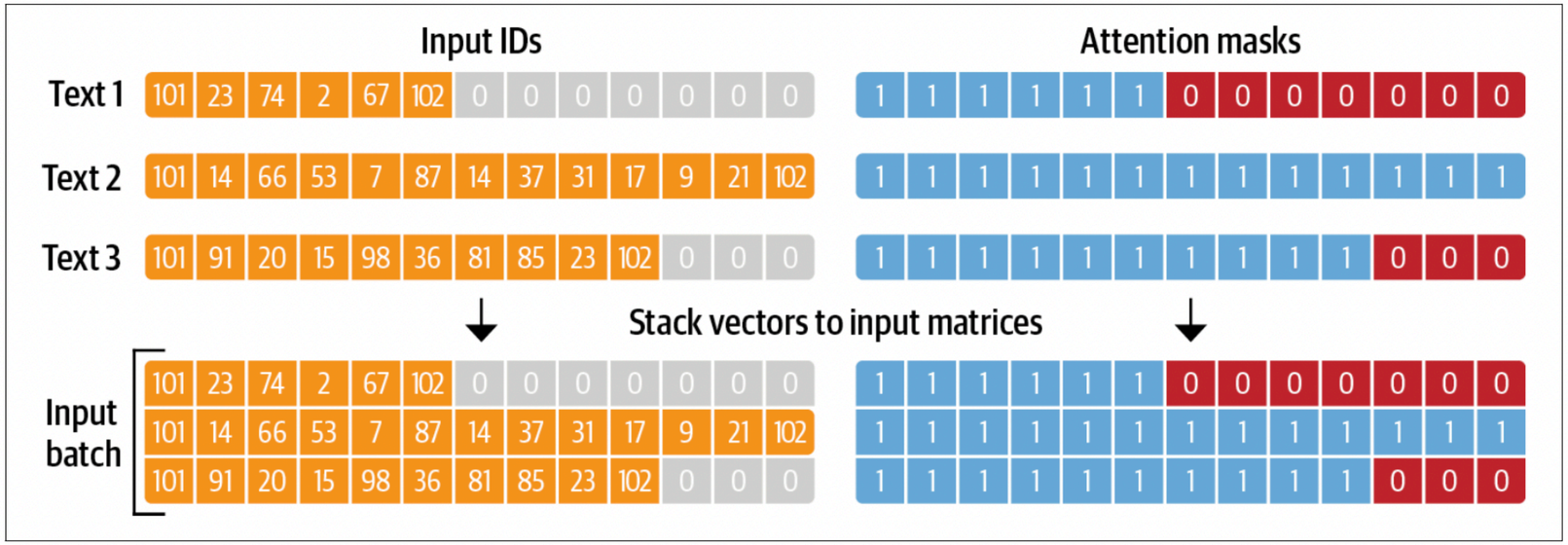


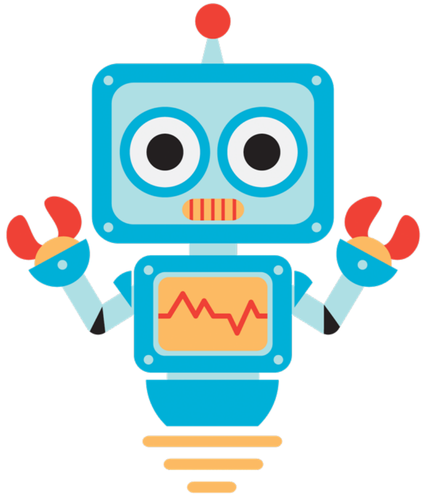
Text classification

From Text to Tokens

Tokenizing the Whole Dataset

Special Token	[PAD]	[UNK]	[CLS]	[SEP]	[MASK]
Special Token ID	0	100	101	102	103





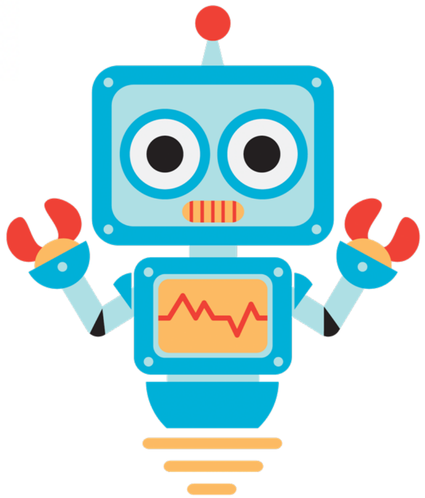
Text classification

From Text to Tokens

Tokenizing the Whole Dataset

Once we've defined a processing function, we can apply it across all the splits in the corpus in a single line of code:

```
emotions_encoded = emotions.map(tokenize, batched=True, batch_size=None)
```



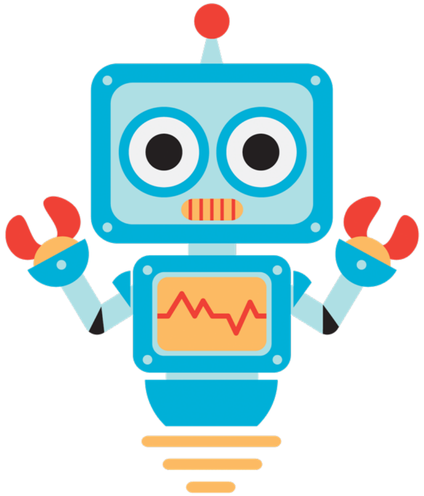
Text classification

From Text to Tokens

Tokenizing the Whole Dataset

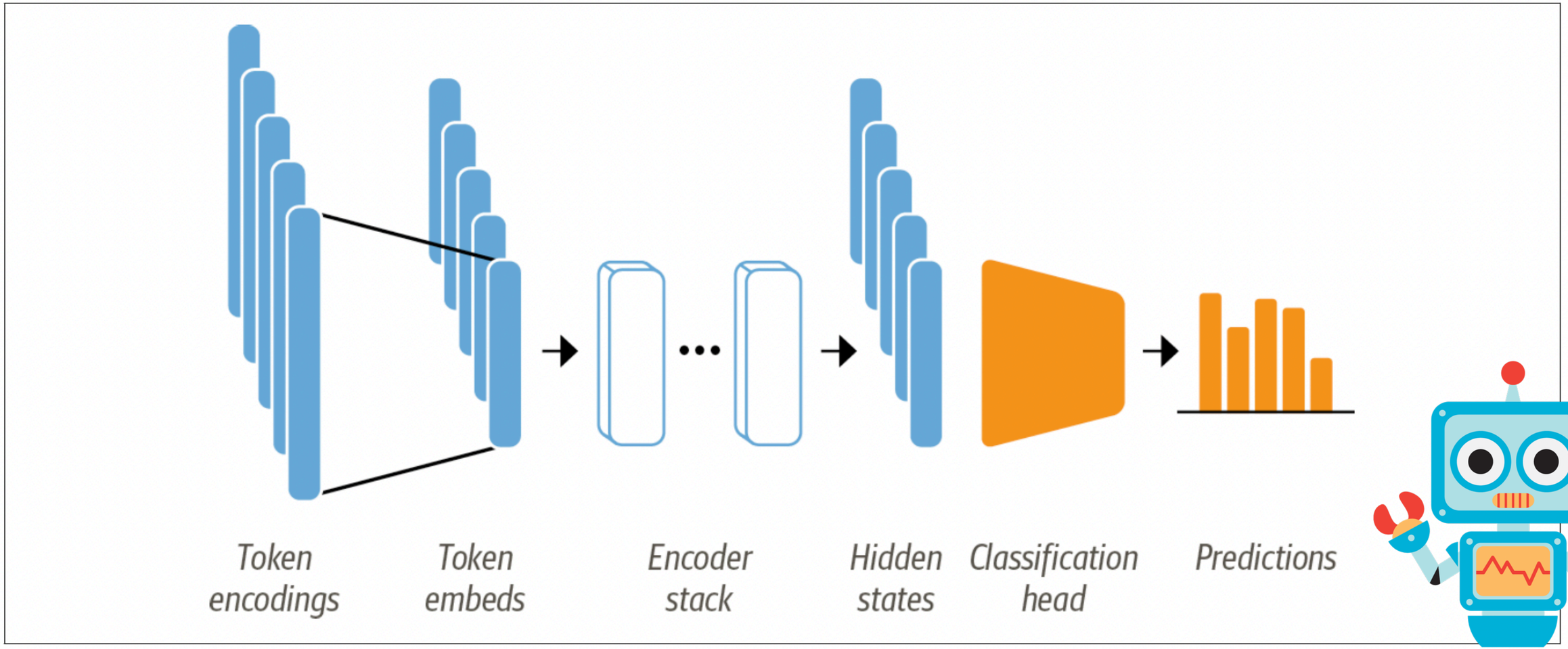
```
emotions_encoded = emotions.map(tokenize, batched=True, batch_size=None)
```

By default, the `map()` method operates individually on every example in the corpus, so setting `batched=True` will encode the tweets in batches. Because we've set `batch_size=None`, our `tokenize()` function will be applied on the full dataset as a single batch. This ensures that the input tensors and attention masks have the same shape globally, and we can see that this operation has added new `input_ids` and `attention_mask` columns to the dataset:



Text classification

Training a Text Classifier



Text classification

Training a Text Classifier

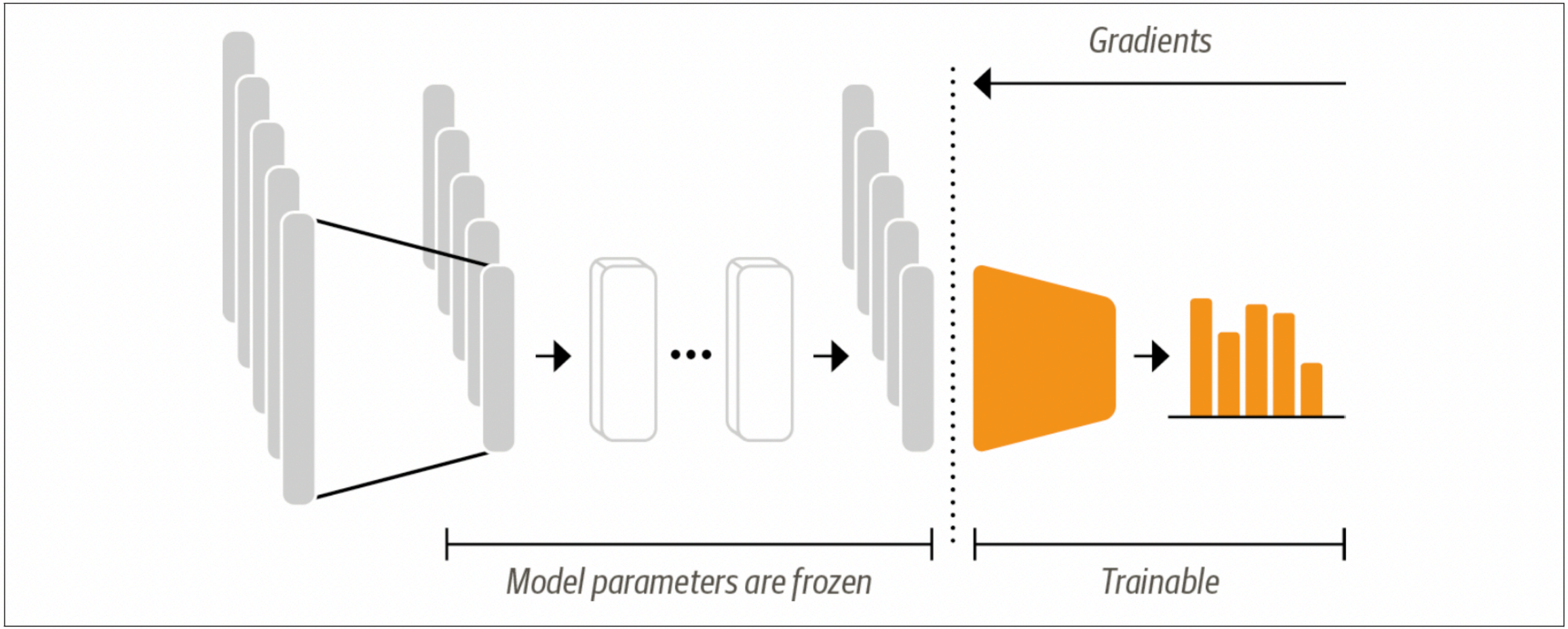
We have two options to train such a model on our Twitter dataset:

Feature extraction
We use the hidden states as features and just train a classifier on them, without modifying the pretrained model.

Fine-tuning
We train the whole model end-to-end, which also updates the parameters of the pretrained model.

Text classification

Feature extraction



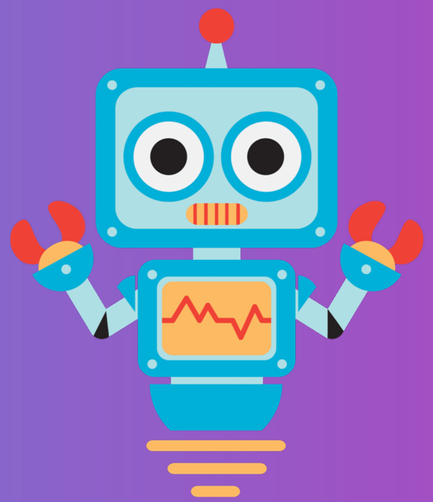
Text classification

Fine-Tuning Transformers

Thank you for your attention

Day 3

January 18th 2024



GAME Khoot