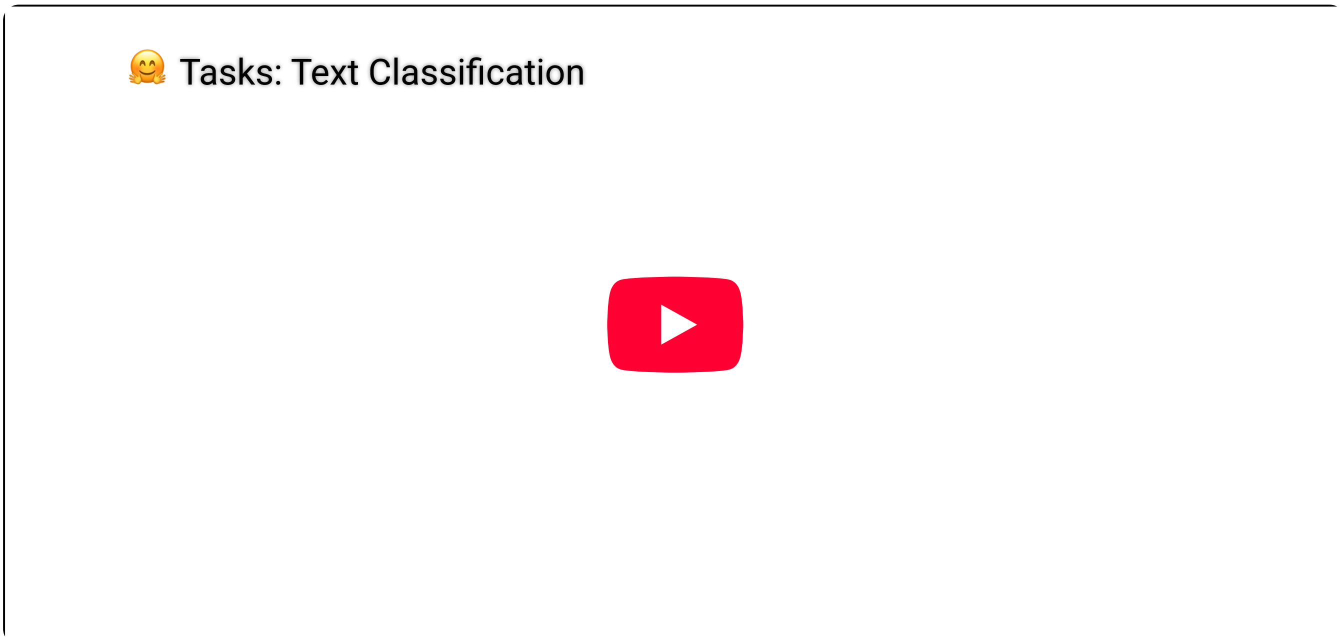
Search models, datasets, users...

Transformers documentation

Text classification

Text classification

Open in Colab Open Studio Lab Copy page



Text classification is a common NLP task that assigns a label or class to text. Some of the largest companies run text classification in production for a wide range of practical applications. One of the most popular forms of text classification is sentiment analysis, which assigns a label like 🙂 positive, 🙁 negative, or 😐 neutral to a sequence of text.

This guide will show you how to:

1. Finetune [DistilBERT](https://huggingface.co/distilbert/distilbert-base-uncased) on the [IMDb](https://huggingface.co/datasets/imdb) dataset to determine whether a movie review is positive or negative.
2. Use your finetuned model for inference.

> To see all architectures and checkpoints compatible with this task, we recommend checking the task-page.

Before you begin, make sure you have all the necessary libraries installed:

```
pip install transformers datasets evaluate accelerate
```

We encourage you to login to your Hugging Face account so you can upload and share your model with the community. When prompted, enter your token to login:

```
>>> from huggingface_hub import notebook_login

>>> notebook_login()
```

Load IMDb dataset

Start by loading the IMDb dataset from the 🤗 Datasets library:

```
>>> from datasets import load_dataset

>>> imdb = load_dataset("imdb")
```

Then take a look at an example:

```
>>> imdb["test"][0]
{
    "label": 0,
    "text": "I love sci-fi and am willing to put up with a lot. Sci-fi movies/TV are u
}
```

There are two fields in this dataset:

- `text`: the movie review text.
- `label`: a value that is either `0` for a negative review or `1` for a positive review.

Preprocess

The next step is to load a DistilBERT tokenizer to preprocess the `text` field:

```
>>> from transformers import AutoTokenizer
```

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

Create a preprocessing function to tokenize `text` and truncate sequences to be no longer than DistilBERT's maximum input length:

```
>>> def preprocess_function(examples):
...     return tokenizer(examples["text"], truncation=True)
```

To apply the preprocessing function over the entire dataset, use 🤗 Datasets map function. You can speed up `map` by setting `batched=True` to process multiple elements of the dataset at once:

```
tokenized_imdb = imdb.map(preprocess_function, batched=True)
```

Now create a batch of examples using DataCollatorWithPadding. It's more efficient to *dynamically pad* the sentences to the longest length in a batch during collation, instead of padding the whole dataset to the maximum length.

```
>>> from transformers import DataCollatorWithPadding

>>> data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

Evaluate

Including a metric during training is often helpful for evaluating your model's performance. You can quickly load a evaluation method with the 🤗 Evaluate library. For this task, load the accuracy metric (see the 🤗 Evaluate quick tour to learn more about how to load and compute a metric):

```
>>> import evaluate

>>> accuracy = evaluate.load("accuracy")
```

Then create a function that passes your predictions and labels to compute to calculate the accuracy:

```
>>> import numpy as np


>>> def compute_metrics(eval_pred):
...     predictions, labels = eval_pred
...     predictions = np.argmax(predictions, axis=1)
...     return accuracy.compute(predictions=predictions, references=labels)
```

Your `compute_metrics` function is ready to go now, and you'll return to it when you setup your training.

Train

Before you start training your model, create a map of the expected ids to their labels with `id2label` and `label2id`:

```
>>> id2label = {0: "NEGATIVE", 1: "POSITIVE"}
>>> label2id = {"NEGATIVE": 0, "POSITIVE": 1}
```

> If you aren't familiar with finetuning a model with the Trainer, take a look at the basic tutorial here!

You're ready to start training your model now! Load DistilBERT with AutoModelForSequenceClassification along with the number of expected labels, and the label mappings:

```
>>> from transformers import AutoModelForSequenceClassification, TrainingArguments, Tr

>>> model = AutoModelForSequenceClassification.from_pretrained(
...     "distilbert/distilbert-base-uncased", num_labels=2, id2label=id2label, label2i
... )
```

At this point, only three steps remain:

1. Define your training hyperparameters in TrainingArguments. The only required parameter is `output_dir` which specifies where to save your model. You'll push this model to the Hub by setting `push_to_hub=True` (you need to be signed in to Hugging Face to upload your

model). At the end of each epoch, the Trainer will evaluate the accuracy and save the training checkpoint.

2. Pass the training arguments to Trainer along with the model, dataset, tokenizer, data collator, and `compute_metrics` function.
3. Call train() to finetune your model.

```
>>> training_args = TrainingArguments(
...     output_dir="my_awesome_model",
...     learning_rate=2e-5,
...     per_device_train_batch_size=16,
...     per_device_eval_batch_size=16,
...     num_train_epochs=2,
...     weight_decay=0.01,
...     eval_strategy="epoch",
...     save_strategy="epoch",
...     load_best_model_at_end=True,
...     push_to_hub=True,
... )

>>> trainer = Trainer(
...     model=model,
...     args=training_args,
...     train_dataset=tokenized_imdb["train"],
...     eval_dataset=tokenized_imdb["test"],
...     processing_class=tokenizer,
...     data_collator=data_collator,
...     compute_metrics=compute_metrics,
... )

>>> trainer.train()
```

> Trainer applies dynamic padding by default when you pass `tokenizer` to it. In this case, you don't need to specify a data collator explicitly.

Once training is completed, share your model to the Hub with the push_to_hub() method so everyone can use your model:

```
>>> trainer.push_to_hub()
```

> For a more in-depth example of how to finetune a model for text classification, take a look at the corresponding PyTorch notebook.

Inference

Great, now that you've finetuned a model, you can use it for inference!

Grab some text you'd like to run inference on:

```
>>> text = "This was a masterpiece. Not completely faithful to the books, but enthrall
```

The simplest way to try out your finetuned model for inference is to use it in a pipeline(). Instantiate a `pipeline` for sentiment analysis with your model, and pass your text to it:

```
>>> from transformers import pipeline

>>> classifier = pipeline("sentiment-analysis", model="stevhliu/my_awesome_model")
>>> classifier(text)
[{'label': 'POSITIVE', 'score': 0.9994940757751465}]
```

You can also manually replicate the results of the `pipeline` if you'd like:

Tokenize the text and return PyTorch tensors:

```
>>> from transformers import AutoTokenizer

>>> tokenizer = AutoTokenizer.from_pretrained("stevhliu/my_awesome_model")
>>> inputs = tokenizer(text, return_tensors="pt")
```

Pass your inputs to the model and return the `logits`:

```
>>> from transformers import AutoModelForSequenceClassification

>>> model = AutoModelForSequenceClassification.from_pretrained("stevhliu/my_awesome_mo
>>> with torch.no_grad():
...     logits = model(**inputs).logits
```

Get the class with the highest probability, and use the model's `id2label` mapping to convert it to a text label:

```
>>> predicted_class_id = logits.argmax().item()
>>> model.config.id2label[predicted_class_id]
'POSITIVE'
```

</> Update on GitHub

← TorchScript Token classification →