```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from datasets import load_dataset

ds = load_dataset('stanfordnlp/imdb')
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as s
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
README.md:       7.81k/? [00:00<00:00, 113kB/s]
train-00000-of-00001.parquet: 100%  21.0M/21.0M [00:00<00:00, 32.7MB/s]
test-00000-of-00001.parquet: 100%  20.5M/20.5M [00:00<00:00, 158MB/s]
unsupervised-00000-of-00001.parquet: 100%  42.0M/42.0M [00:00<00:00, 186MB/s]
Generating train split: 100%  25000/25000 [00:00<00:00, 70202.42 examples/s]
Generating test split: 100%  25000/25000 [00:00<00:00, 99791.91 examples/s]
Generating unsupervised split: 100%  50000/50000 [00:00<00:00, 90182.24 examples/s]
```

```
ds
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 25000
    })
    unsupervised: Dataset({
        features: ['text', 'label'],
        num_rows: 50000
    })
})
```

```
from transformers import AutoTokenizer

checkpoint = 'bert-base-uncased'
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
```

```
tokenizer_config.json: 100%  48.0/48.0 [00:00<00:00, 4.26kB/s]
config.json: 100%  570/570 [00:00<00:00, 47.8kB/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 2.11MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 8.24MB/s]
```

```
sample = ds['train'][5]
sample
```

```
{'text': "I would put this at the top of my list of films in the category of unwatchable trash! There are films that are bad, but
the worst kind are the ones that are unwatchable but you are suppose to like them because they are supposed to be good for you! The
sex sequences, so shocking in its day, couldn't even arouse a rabbit. The so called controversial politics is strictly high school
sophomore amateur night Marxism. The film is self-consciously arty in the worst sense of the term. The photography is in a harsh
grainy black and white. Some scenes are out of focus or taken from the wrong angle. Even the sound is bad! And some people call
this art?<br /><br />",
 'label': 0}
```

```
tokenizer(sample['text'])
```

```
{'input_ids': [101, 1045, 2052, 2404, 2023, 2012, 1996, 2327, 1997, 2026, 2862, 1997, 3152, 1999, 1996, 4696, 1997, 4895, 18866,
3085, 11669, 999, 2045, 2024, 3152, 2008, 2024, 2919, 1010, 2021, 1996, 5409, 2785, 2024, 1996, 3924, 2008, 2024, 4895, 18866,
3085, 2021, 2017, 2024, 6814, 2000, 2066, 2068, 2138, 2027, 2024, 4011, 2000, 2022, 2204, 2005, 2017, 999, 1996, 3348, 10071, 1010,
2061, 16880, 1999, 2049, 2154, 1010, 2481, 1005, 1056, 2130, 12098, 15441, 1037, 10442, 1012, 1996, 2061, 2170, 6801, 4331, 2003,
9975, 2152, 2082, 13758, 5515, 2305, 27255, 1012, 1996, 2143, 2003, 2969, 1011, 24447, 2396, 2100, 1999, 1996, 5409, 3168, 1997,
1996, 2744, 1012, 1996, 5855, 2003, 1999, 1037, 8401, 8982, 2100, 2304, 1998, 2317, 1012, 2070, 5019, 2024, 2041, 1997, 3579, 2030,
2579, 2013, 1996, 3308, 6466, 1012, 2130, 1996, 2614, 2003, 2919, 999, 1998, 2070, 2111, 2655, 2023, 2396, 1029, 1026, 7987, 1013,
1028, 1026, 7987, 1013, 1028, 102], 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

```python
def tokenize_function(example):
  return tokenizer(example['text'], truncation=True)

tokenized_datasets = ds.map(tokenize_function, batched=True)
tokenized_datasets
```

```
Map: 100%    25000/25000 [00:25<00:00, 1082.22 examples/s]
Map: 100%    25000/25000 [00:25<00:00, 796.29 examples/s]
Map: 100%    50000/50000 [00:49<00:00, 970.54 examples/s]
DatasetDict({
    train: Dataset({
        features: ['text', 'label', 'input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 25000
    })
    test: Dataset({
        features: ['text', 'label', 'input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 25000
    })
    unsupervised: Dataset({
        features: ['text', 'label', 'input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 50000
    })
})
```

```python
from transformers import DataCollatorWithPadding

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

```python
from transformers import TrainingArguments

training_args = TrainingArguments('test_trainer', eval_strategy='epoch', report_to="none")
```

```python
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)
```

```
model.safetensors: 100%    440M/440M [00:08<00:00, 64.3MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly ini1
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```python
from sklearn.metrics import f1_score, accuracy_score
import numpy as np

def compute_metrics(eval_preds):
  logits, labels = eval_preds
  predictions = np.argmax(logits, axis=-1)
  return {'accuracy': accuracy_score(predictions, labels), 'f1': f1_score(predictions, labels)}
```

```python
from transformers import Trainer

trainer = Trainer(model,
                  training_args,
                  train_dataset=tokenized_datasets['train'],
                  eval_dataset=tokenized_datasets['test'],
                  data_collator=data_collator,
                  processing_class=tokenizer,
                  compute_metrics=compute_metrics)
```

```python
trainer.train()
```

[9375/9375 2:30:20, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	0.295200	0.432647	0.870760	0.855727
2	0.183500	0.273526	0.929960	0.928152
3	0.082400	0.312541	0.937200	0.937435

```
TrainOutput(global_step=9375, training_loss=0.20250566833496095, metrics={'train_runtime': 9022.7938, 'train_samples_per_second':
8.312, 'train_steps_per_second': 1.039, 'total_flos': 1.860035650317888e+16, 'train_loss': 0.20250566833496095, 'epoch': 3.0})
```

```
output_dir = "/content/drive/MyDrive/imdb_model_final"
trainer.save_model(output_dir)
tokenizer.save_pretrained(output_dir)
print(f"Модель сохранена в {output_dir}")
```

```
Модель сохранена в /content/drive/MyDrive/imdb_model_final
```

```
model_path = "/content/drive/MyDrive/imdb_model_final"

model = AutoModelForSequenceClassification.from_pretrained(model_path)
tokenizer = AutoTokenizer.from_pretrained(model_path)

print("Модель и токенайзер загружены!")
```

```
Модель и токенайзер загружены!
```

```
import torch

def predict(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True)

    with torch.no_grad():
        outputs = model(**inputs)

    probs = torch.nn.functional.softmax(outputs.logits, dim=-1)

    predicted_class = torch.argmax(probs, dim=-1).item()
    confidence = probs[0][predicted_class].item()

    sentiment = "Позитивный" if predicted_class == 1 else "Негативный"
    return sentiment, confidence


test_texts = [
    "This movie was fantastic! I really loved it.",
    "The acting was terrible and the plot made no sense.",
    "It was okay, nothing special.",
    "I have never seen such a brilliant film in my life!",
    "This is the worst movie I've ever watched."
]

for text in test_texts:
    sentiment, confidence = predict(text)
    print(f"Текст: {text}")
    print(f"Сентимент: {sentiment} (уверенность: {confidence:.2f})")
    print("-" * 80)
```

```
Текст: This movie was fantastic! I really loved it.
Сентимент: Позитивный (уверенность: 1.00)
--------------------------------------------------------------------------------
Текст: The acting was terrible and the plot made no sense.
Сентимент: Негативный (уверенность: 1.00)
--------------------------------------------------------------------------------
Текст: It was okay, nothing special.
Сентимент: Негативный (уверенность: 1.00)
--------------------------------------------------------------------------------
Текст: I have never seen such a brilliant film in my life!
Сентимент: Позитивный (уверенность: 1.00)
--------------------------------------------------------------------------------
Текст: This is the worst movie I've ever watched.
Сентимент: Негативный (уверенность: 1.00)
--------------------------------------------------------------------------------
```

```
def predict(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True, max_length=512)
    with torch.no_grad():
        outputs = model(**inputs)
    predicted_class = torch.argmax(outputs.logits, dim=-1).item()
    return predicted_class

test_dataset = ds['test']
```

```
errors = []
for i, example in enumerate(test_dataset):
    text = example['text']
    true_label = example['label']
    predicted_label = predict(text)

    if true_label != predicted_label:
        errors.append({
            'text': text,
            'true_label': true_label,
            'predicted_label': predicted_label
        })

    if len(errors) >= 5:
        break

for i, error in enumerate(errors):
    print(f"Пример {i+1}:")
    print(f"Истинный сентимент: {'Позитивный' if error['true_label'] == 1 else 'Негативный'}")
    print(f"Предсказанный сентимент: {'Позитивный' if error['predicted_label'] == 1 else 'Негативный'}")
    print(f"Текст: {error['text']}")
    print("-" * 100)
```

```
Пример 1:
Истинный сентимент: Негативный
Предсказанный сентимент: Позитивный
Текст: First off let me say, If you haven't enjoyed a Van Damme movie since bloodsport, you probably will not like this movie. Most
----------------------------------------------------------------------------------------------------
Пример 2:
Истинный сентимент: Негативный
Предсказанный сентимент: Позитивный
Текст: Low budget horror movie. If you don't raise your expectations too high, you'll probably enjoy this little flick. Beginning ar
----------------------------------------------------------------------------------------------------
Пример 3:
Истинный сентимент: Негативный
Предсказанный сентимент: Позитивный
Текст: I'm the type of guy who loves hood movies from New Jack City to Baby Boy to Killa Season, from the b grade to the Hollywood.
----------------------------------------------------------------------------------------------------
Пример 4:
Истинный сентимент: Негативный
Предсказанный сентимент: Позитивный
Текст: An obscure horror show filmed in the Everglades. Two couples stay overnight in a cabin after being made a little uneasy by th
----------------------------------------------------------------------------------------------------
Пример 5:
Истинный сентимент: Негативный
Предсказанный сентимент: Позитивный
Текст: 1983's "Frightmare" is an odd little film. The director seems to be trying to combine the atmosphere of classic '30s/'40s sty
----------------------------------------------------------------------------------------------------
```