Quá trình

Bộ dữ liệu cung cấp nhiều công cụ để sửa đổi cấu trúc và nội dung của bộ dữ liệu. Những cái này công cụ rất quan trọng để sắp xếp tập dữ liệu, tạo các cột bổ sung, chuyển đổi giữa các tính năng và định dạng, và nhiều hơn nữa.

Hướng dẫn này sẽ chỉ cho bạn cách:

- Sắp xếp lại các hàng và chia tập dữ liệu.
- Đổi tên và xóa các cột cũng như các thao tác cột thông thường khác.
- Áp dụng các hàm xử lý cho từng mẫu trong tập dữ liệu.
- Ghép nối các tập dữ liệu.
- Áp dụng một phép biến đổi định dạng tùy chỉnh.
- Lưu và xuất các tập dữ liệu đã xử lý.

Để biết thêm chi tiết cụ thể về việc xử lý các phương thức tập dữ liệu khác, hãy xem quy trình hướng dẫn tập dữ liệu âm thanh, hướng dẫn tập dữ liệu hình ảnh quy trình hoặc hướng dẫn tập dữ liệu văn

Các ví dụ trong hướng dẫn này sử dụng tập dữ liệu MRPC, nhưng vui lòng tải bất kỳ tập dữ liệu nào của bạn hãy lựa chọn và làm theo nhé!

```
>>> from datasets import load_dataset
>>> dataset = load_dataset("nyu-mll/glue", "mrpc", split="train")
```

[!WARNING]
Tất cả các phương pháp xử lý trong hướng dẫn này đều trả về một đối tượng Bộ dữ liệu mới. Sửa đổi khôn tại chỗ. Hãy cẩn thận về việc ghi đè tập dữ liệu trước đó của bạn!

Sắp xếp, xáo trộn, chọn, tách và phân đoạn

Có một số chức năng để sắp xếp lại cấu trúc của tập dữ liệu.
Các hàm này hữu ích khi chỉ chọn các hàng bạn muốn, tạo các phần tách đào tạo và kiểm tra, và chia nhỏ các tập dữ liệu rất lớn thành các phần nhỏ hơn.

Loại

Use sort() to sort column values according to their numerical values. The provided column
phải tương thích với NumPy.

```
>>> dataset["label"][:10]
[1, 0, 1, 0, 1, 1, 0, 1, 0, 0]
>>> sorted_dataset = dataset.sort("label")
>>> sorted_dataset["label"][:10]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
>>> sorted_dataset["label"][-10:]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

Dưới mui xe, điều này tạo ra một danh sách các chỉ mục được sắp xếp theo giá trị của cột.
Ánh xạ chỉ mục này sau đó được sử dụng để truy cập vào các hàng bên phải trong bảng Mũi tên bên dưới.

Trộn bài

The shuffle() function randomly rearranges the column values. You can specify the generator
tham số trong hàm này để sử dụng một numpy.random.Generator khác nếu bạn muốn kiểm soát nhiều hơn
về thuật toán được sử dụng để xáo trộn tập dữ liệu.

```
>>> shuffled_dataset = sorted_dataset.shuffle(seed=42)
>>> shuffled_dataset["label"][:10]
[1, 1, 1, 0, 1, 1, 1, 1, 1, 0]
```

Shuffling takes the list of indices [0:len(my_dataset)] and shuffles it to create an indices
lập bản đồ.
Tuy nhiên, ngay sau khi Tập dữ liệu của bạn có ánh xạ chỉ mục, tốc độ có thể chậm hơn gấp 10 lần.
Điều này là do có thêm một bước để đọc chỉ mục hàng bằng cách sử dụng ánh xạ chỉ mục,
và quan trọng nhất, bạn không còn đọc những khối dữ liệu liền kề nhau nữa.
Để khôi phục tốc độ, bạn cần phải ghi lại toàn bộ tập dữ liệu trên đĩa của mình bằng cách sử dụng
Dataset.flatten_indices(), which removes the indices mapping.
Ngoài ra, bạn có thể chuyển sang IterableDataset và tận dụng khả năng xáo trộn gần đúng nhanh chóng của
IterableDataset.shuffle():

```
>>> iterable_dataset = dataset.to_iterable_dataset(num_shards=128)
>>> shuffled_iterable_dataset = iterable_dataset.shuffle(seed=42, buffer_size=1000)
```

Chọn và lọc

There are two options for filtering rows in a dataset: select() and filter().

- select() returns rows according to a list of indices:

```
>>> small_dataset = dataset.select([0, 10, 20, 30, 40, 50])
>>> len(small_dataset)
6
```

- filter() returns rows that match a specified condition:

```
>>> start_with_ar = dataset.filter(lambda example: example["sentence1"].startswith("Ar"))
>>> len(start_with_ar)
6
>>> start_with_ar["sentence1"]
['Around 0335 GMT , Tab shares were up 19 cents , or 4.4 % , at A $ 4.56 , having earlier set a re
'Arison cho biết Mann có thể là một trong những người tiên phong của phong trào âm nhạc thế giới và anh
'Arts đã giúp huấn luyện thanh thiếu niên trong đội bóng đá lớp 8 tại Trường Trung học cơ sở Lombardi ở C
'Around 9 : 00 a.m. EDT ( 1300 GMT ) , the euro was at $ 1.1566 against the dollar , up 0.07 perce
"Cho rằng vụ việc chỉ là một ví dụ cá biệt , Canada đã đe dọa sẽ có phản ứng dữ dội về mặt thương mại nếu
'Các nghệ sĩ lo lắng kế hoạch này sẽ gây tổn hại cho những người cần giúp đỡ nhất – những người biểu diễ
]
```

filter() can also filter by indices if you set with_indices=True :

```
>>> even_dataset = dataset.filter(lambda example, idx: idx % 2 == 0, with_indices=True)
>>> len(even_dataset)
1834
>>> len(dataset) / 2
1834.0
```

Trừ khi danh sách các chỉ mục cần giữ liền kề nhau, các phương thức đó cũng tạo ra ánh xạ chỉ mục dưới mui xe.

Tách ra

The train_test_split() function creates train and test splits if your dataset doesn't already have họ. Điều này cho phép bạn điều chỉnh tỷ lệ tương đối hoặc số lượng mẫu tuyệt đối trong mỗi lần chia tay. Trong ví dụ bên dưới, hãy sử dụng tham số test_size để tạo phần tách thử nghiệm 10% của tập dữ liệu gốc:

```
>>> dataset.train_test_split(test_size=0.1)
{'train': Dataset(schema: {'sentence1': 'string', 'sentence2': 'string', 'label': 'int64', 'idx':
'test': Dataset(schema: {'sentence1': 'string', 'sentence2': 'string', 'label': 'int64', 'idx': 'i
>>> 0.1 * len(dataset)
366,8
```

The splits are shuffled by default, but you can set shuffle=False to prevent shuffling.

Mảnh vỡ

Bộ dữ liệu hỗ trợ phân đoạn để chia một tập dữ liệu rất lớn thành một số lượng được xác định trước chunks. Specify the num_shards parameter in shard() to determine the number of shards to chia tập dữ liệu thành. Bạn cũng cần cung cấp phân đoạn mà bạn muốn trả về cùng với chỉ mục tham số.

Ví dụ: tập dữ liệu stanfordnlp/imdb có 25000 ví dụ:

```
>>> from datasets import load_dataset
>>> dataset = load_dataset("stanfordnlp/imdb", split="train")
>>> print(dataset)
Dataset({
    features: ['text', 'label'],
    num_rows: 25000
})
```

Sau khi chia tập dữ liệu thành bốn phần, phân đoạn đầu tiên sẽ chỉ có 6250 ví dụ:

```
>>> dataset.shard(num_shards=4, index=0)
Dataset({
    features: ['text', 'label'],
    num_rows: 6250
})
>>> print(25000/4)
6250.0
```

Đổi tên, xóa, truyền và làm phẳng

Các hàm sau cho phép bạn sửa đổi các cột của tập dữ liệu. Các chức năng này được hữu ích cho việc đổi tên hoặc xóa cột, thay đổi cột thành một tập hợp tính năng mới và làm phẳng các cấu trúc cột lồng nhau.

Đổi tên

Use rename_column() when you need to rename a column in your dataset. Features liên kết với cột ban đầu thực sự được di chuyển dưới tên cột mới, thay vì chỉ cần thay thế cột ban đầu tại chỗ.

Provide rename_column() with the name of the original column, and the new column name:

```
>>> dataset
Dataset({
    features: ['sentence1', 'sentence2', 'label', 'idx'],
    num_rows: 3668
})
>>> dataset = dataset.rename_column("sentence1", "sentenceA")
>>> dataset = dataset.rename_column("sentence2", "sentenceB")
>>> dataset
Dataset({
    features: ['sentenceA', 'sentenceB', 'label', 'idx'],
    num_rows: 3668
})
```

Di dời

Khi cần xóa một hoặc nhiều cột, hãy cung cấp tên cột cần xóa cho remove_columns() function. Remove more than one column by providing a list of column tên:

```
>>> dataset = dataset.remove_columns("label")
>>> dataset
Dataset({
    features: ['sentence1', 'sentence2', 'idx'],
    num_rows: 3668
})
>>> dataset = dataset.remove_columns(["sentence1", "sentence2"])
>>> dataset
Dataset({
    features: ['idx'],
    num_rows: 3668
})
```

Conversely, select_columns() selects one or more columns to keep and removes the rest. This hàm lấy một hoặc một danh sách tên cột:

```
>>> dataset
Dataset({
    features: ['sentence1', 'sentence2', 'label', 'idx'],
    num_rows: 3668
})
>>> dataset = dataset.select_columns(['sentence1', 'sentence2', 'idx'])
>>> dataset
Dataset({
    features: ['sentence1', 'sentence2', 'idx'],
    num_rows: 3668
})
>>> dataset = dataset.select_columns('idx')
>>> dataset
Dataset({
    features: ['idx'],
    num_rows: 3668
})
```

Dàn diễn viên

The cast() function transforms the feature type of one or more columns. This function accepts
Tính năng mới của bạn làm đối số. Ví dụ dưới đây minh họa cách thay đổi
Các tính năng của ClassLabel và Value:

```
>>> dataset.features
{'sentence1': Value('string'),
'sentence2': Value('string'),
'label': ClassLabel(names=['not_equivalent', 'equivalent']),
'idx': Value('int32')}

>>> from datasets import ClassLabel, Value
>>> new_features = dataset.features.copy()
>>> new_features["label"] = ClassLabel(names=["negative", "positive"])
>>> new_features["idx"] = Value("int64")
>>> dataset = dataset.cast(new_features)
>>> dataset.features
{'sentence1': Value('string'),
'sentence2': Value('string'),
'label': ClassLabel(names=['negative', 'positive']),
'idx': Value('int64')}
```

[!TIP]
Truyền chỉ hoạt động nếu loại tính năng ban đầu và loại tính năng mới tương thích. Vì example, you can cast a column with the feature type Value("int32") to Value("bool") if cột ban đầu chỉ chứa số một và số không.

Use the cast_column() function to change the feature type of a single column. Pass the column tên và loại tính năng mới của nó làm đối số:

```
>>> dataset.features
{'audio': Audio(sampling_rate=44100, mono=True)}

>>> dataset = dataset.cast_column("audio", Audio(sampling_rate=16000))
>>> dataset.features
{'audio': Audio(sampling_rate=16000, mono=True)}
```

Làm phẳng

Đôi khi một cột có thể là một cấu trúc lồng nhau của nhiều loại. Hãy nhìn vào lồng nhau cấu trúc bên dưới từ bộ dữ liệu SQuAD:

```
>>> from datasets import load_dataset
>>> dataset = load_dataset("rajpurkar/squad", split="train")
>>> dataset.features
{'id': Value('string'),
 'title': Value('string'),
 'context': Value('string'),
 'question': Value('string'),
 'answers': {'text': List(Value('string')),
  'answer_start': List(Value('int32'))}}
```

The answers field contains two subfields: text and answer_start . Use the flatten() function để trích xuất các trường con thành các cột riêng biệt của chúng:

```
>>> flat_dataset = dataset.flatten()
>>> flat_dataset
Dataset({
      features: ['id', 'title', 'context', 'question', 'answers.text', 'answers.answer_start'],
 num_rows: 87599
})
```

Lưu ý rằng các trường con bây giờ là các cột độc lập của riêng chúng: câu trả lời.text và câu trả lời.answer_start .

Bản đồ

Some of the more powerful applications of Datasets come from using the map() function. The primary purpose of map() is to speed up processing functions. It allows you to apply a chức năng xử lý cho từng mẫu trong tập dữ liệu, độc lập hoặc theo đợt. Chức năng này thậm chí có thể tạo hàng và cột mới.

Trong ví dụ sau, hãy thêm 'Câu của tôi:' vào tiền tố mỗi giá trị câu1 trong tập dữ liệu.

Bắt đầu bằng cách tạo một hàm thêm 'Câu của tôi:' vào đầu mỗi câu. các chức năng cần chấp nhận và xuất ra một lệnh:

```
>>> def add_prefix(example):
...example["sentence1"] = 'My sentence: ' + example["sentence1"]
...trả lại ví dụ
```

Now use map() to apply the add_prefix function to the entire dataset:

```
>>> updated_dataset = small_dataset.map(add_prefix)
>>> updated_dataset["sentence1"][:5]
['My sentence: Amrozi accused his brother , whom he called " the witness " , of deliberately disto
"Câu của tôi: Yucaipa sở hữu Dominick's trước khi bán chuỗi này cho Safeway vào năm 1998 với giá 2,5 đô
'Câu của tôi: Họ đã đăng một quảng cáo trên Internet vào ngày 10 tháng 6, cung cấp hàng hóa
'Câu của tôi: Vào khoảng 03:35 GMT, cổ phiếu của Tab đã tăng 19 xu, hay 4,4 %, ở mức 4,56 đô la Úc, có bá
]
```

Let's take a look at another example, except this time, you'll remove a column with map(). Khi bạn xóa một cột, nó chỉ bị xóa sau khi ví dụ đã được cung cấp cho chức năng ánh xạ. Điều này cho phép hàm ánh xạ sử dụng nội dung của các cột trước chúng được loại bỏ.

Specify the column to remove with the remove_columns parameter in map():

```
>>> updated_dataset = dataset.map(lambda example: {"new_sentence": example["sentence1"]}, remove_c
>>> updated_dataset.column_names
['sentence2', 'label', 'idx', 'new_sentence']
```

[!TIP]
Datasets also has a remove_columns() function which is faster because it doesn't copy dữ liệu của các cột còn lại.

You can also use map() with indices if you set with_indices=True . The example below adds chỉ mục ở đầu mỗi câu:

```
>>> updated_dataset = dataset.map(lambda example, idx: {"sentence2": f"{idx}: " + example["sentenc
>>> updated_dataset["sentence2"][:5]
['0: Referring to him as only " the witness " , Amrozi accused his brother of deliberately distort
 "1: Yucaipa mua Dominick's vào năm 1995 với giá 693 triệu USD và bán lại cho Safeway với giá 1,8 tỷ USD
 "2: Vào ngày 10 tháng 6, các chủ tàu đã đăng một quảng cáo trên Internet, chào mời
 '3: Cổ phiếu tab đã tăng 20 xu , hay 4,6 % , để lập mức đóng cửa cao kỷ lục ở mức 4,57 đô la Úc .',
 '4: Cổ phiếu của PG & E Corp. đã tăng 1,63 USD hay 8% lên 21,03 USD trên Sở giao dịch chứng khoán New
]
```

Đa xử lý

Đa xử lý tăng tốc đáng kể việc xử lý bằng cách song song hóa các quy trình trên CPU. Bộ
the num_proc parameter in map() to set the number of processes to use:

```
>>> updated_dataset = dataset.map(lambda example, idx: {"sentence2": f"{idx}: " + example["sentenc
```

The map() also works with the rank of the process if you set with_rank=True . This is
tương tự với tham số with_indices. Tham số with_rank trong hàm được ánh xạ
theo sau chỉ mục một nếu nó đã có sẵn.

```
>>> import torch
>>> from multiprocess import set_start_method
>>> from transformers import AutoTokenizer, AutoModelForCausalLM
>>> from datasets import load_dataset
>>>
>>> # Get an example dataset
>>> dataset = load_dataset("fka/awesome-chatgpt-prompts", split="train")
>>>
>>> # Get an example model and its tokenizer
>>> model = AutoModelForCausalLM.from_pretrained("Qwen/Qwen1.5-0.5B-Chat").eval()
>>> tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen1.5-0.5B-Chat")
>>>
>>> def gpu_computation(batch, rank):
...# Di chuyển mô hình sang GPU phù hợp nếu chưa có
...device = f"cuda:{(rank or 0) % torch.cuda.device_count()}"
...model.to(device)
None
...# Lệnh gọi GPU lớn của bạn sẽ diễn ra ở đây, ví dụ:
...chats = [[
...{"role": "system", "content": "You are a helpful assistant."},
...{"role": "user", "content": prompt}
...] for prompt in batch["prompt"]]
...texts = [tokenizer.apply_chat_template(
...trò chuyện,
...tokenize=False,
...add_generation_prompt=True
...) for chat in chats]
...model_inputs = tokenizer(texts, padding=True, return_tensors="pt").to(device)
...with torch.no_grad():
...outputs = model.generate(**model_inputs, max_new_tokens=512)
...batch["output"] = tokenizer.batch_decode(outputs, skip_special_tokens=True)
...lô hàng trả lại
>>>
>>> if __name__ == "__main__":
...set_start_method("spawn")
...updated_dataset = dataset.map(
...tính toán gpu_computing,
...batched=True,
...batch_size=16,
```

```
...with_rank=True,
...num_proc=torch.cuda.device_count(),# one process per GPU
...)
```

Trường hợp sử dụng chính của xếp hạng là tính toán song song trên một số GPU. Điều này đòi hỏi setting multiprocess.set_start_method("spawn") . If you don't you'll receive the following CUDA lỗi:

```
RuntimeError: Không thể khởi tạo lại CUDA trong quy trình con rẽ nhánh. Để sử dụng CUDA với đa xử lý, yo
```

Xử lý hàng loạt

The map() function supports working with batches of examples. Operate on batches by setting batched=True . The default batch size is 1000, but you can adjust it with the batch_size tham số. Xử lý hàng loạt cho phép các ứng dụng thú vị như tách các câu dài thành các đoạn ngắn hơn và tăng cường dữ liệu.

Chia các ví dụ dài

Khi các ví dụ quá dài, bạn có thể muốn chia chúng thành nhiều phần nhỏ hơn. Bắt đầu bằng tạo ra một chức năng:

1. Chia trường câu 1 thành các đoạn 50 ký tự.
2. Xếp chồng tất cả các phần lại với nhau để tạo tập dữ liệu mới.

```
>>> def chunk_examples(examples):
...chunks = []
...for sentence in examples["sentence1"]:
...chunks += [sentence[i:i + 50] for i in range(0, len(sentence), 50)]
...return {"chunks": chunks}
```

Apply the function with map():

```
>>> chunked_dataset = dataset.map(chunk_examples, batched=True, remove_columns=dataset.column_na
>>> chunked_dataset[:10]
{'chunks': ['Amrozi accused his brother , whom he called " the ',
            'nhân chứng', về việc cố tình bóp méo bằng chứng của mình',
            'e.',
            "Yucaipa sở hữu Dominick's trước khi bán chuỗi",
            ' tới Safeway năm 1998 với giá 2,5 tỷ USD .',
            'Họ đã xuất bản một quảng cáo trên Internet',
            'Không phải vào ngày 10 tháng 6, anh ta rao bán hàng hóa',
            'ded .',
            'Khoảng 03:35 GMT , cổ phiếu của Tab đã tăng 19 xu , hoặc',
            ' 4.4 % , at A $ 4.56 , having earlier set a record']}
```

Hãy chú ý cách các câu được chia thành các đoạn ngắn hơn và có nhiều hàng hơn trong phần này.
tập dữ liệu.

```
>>> dataset
Dataset({
  features: ['sentence1', 'sentence2', 'label', 'idx'],
  num_rows: 3668
})
>>> chunked_dataset
Dataset({
      features: ['chunks'],
      num_rows: 10470
})
```

Tăng cường dữ liệu

The map() function could also be used for data augmentation. The following example
tạo ra các từ bổ sung cho mã thông báo được che giấu trong một câu.

Tải và sử dụng mô hình RoBERTA trong FillMaskPipeline của Transformers:

```
>>> from random import randint
>>> from transformers import pipeline

>>> fillmask = pipeline("fill-mask", model="roberta-base")
>>> mask_token = fillmask.tokenizer.mask_token
>>> smaller_dataset = dataset.filter(lambda e, i: i<100, with_indices=True)
```

Tạo một hàm để chọn ngẫu nhiên một từ để che dấu trong câu. Chức năng này cũng nên trả lại câu gốc và hai câu thay thế hàng đầu do RoBERTA tạo ra.

```
>>> def augment_data(examples):
...outputs = []
...for sentence in examples["sentence1"]:
...words = sentence.split(' ')
...K = randint(1, len(words)-1)
...masked_sentence = " ".join(words[:K]+ [mask_token] + words[K+1:])
...predictions = fillmask(masked_sentence)
...augmented_sequences = [predictions[i]["sequence"] for i in range(3)]
...outputs += [sentence] + augmented_sequences
None
...return {"data": outputs}
```

Use map() to apply the function over the whole dataset:

```
>>> augmented_dataset = smaller_dataset.map(augment_data, batched=True, remove_columns=dataset.c
>>> augmented_dataset[:9]["data"]
['Amrozi accused his brother , whom he called " the witness " , of deliberately distorting his evi
 'Amrozi buộc tội anh trai mình, người mà anh gọi là "nhân chứng", đã cố tình giấu nhẹm lời khai của mình
 'Amrozi cáo buộc anh trai mình, người mà anh gọi là "nhân chứng", đã cố tình che giấu bằng chứng của n
 'Amrozi buộc tội anh trai mình, người mà anh ta gọi là "nhân chứng", đã cố tình tiêu hủy bằng chứng của
 "Yucaipa sở hữu Dominick's trước khi bán chuỗi này cho Safeway vào năm 1998 với giá 2,5 tỷ USD.",
 'Yucaipa sở hữu Dominick Stores trước khi bán chuỗi này cho Safeway vào năm 1998 với giá 2,5 tỷ USD.',
 "Yucaipa sở hữu Dominick's trước khi bán chuỗi này cho Safeway vào năm 1998 với giá 2,5 tỷ USD.",
 “Yucaipa sở hữu Dominick Pizza trước khi bán chuỗi này cho Safeway vào năm 1998 với giá 2,5 tỷ USD”.
]
```

Đối với mỗi câu gốc, RoBERTA tăng cường một từ ngẫu nhiên bằng ba từ thay thế. các

sự bóp méo từ gốc được bổ sung bằng cách giữ lại, đàn áp và phá hủy.

Xử lý không đồng bộ

Các hàm không đồng bộ rất hữu ích để gọi các điểm cuối API song song, chẳng hạn như để tải xuống nội dung như hình ảnh hoặc gọi điểm cuối mô hình.

Bạn có thể xác định hàm không đồng bộ bằng cách sử dụng từ khóa async và chờ đợi, đây là một hàm ví dụ để gọi mô hình trò chuyện từ Ôm mặt:

```
>>> import aiohttp
>>> import asyncio
>>> from huggingface_hub import get_token
>>> sem = asyncio.Semaphore(20)# max number of simultaneous queries
>>> async def query_model(model, prompt):
...api_url = f"https://api-inference.huggingface.co/models/{model}/v1/chat/completions"
...headers = {"Authorization": f"Bearer {get_token()}", "Content-Type": "application/json"}
...json = {"messages": [{"role": "user", "content": prompt}], "max_tokens": 20, "seed": 42}
...async with sem, aiohttp.ClientSession() as session, session.post(api_url, headers=headers,
...output = await response.json()
...return {"Output": output["choices"][0]["message"]["content"]}
```

Các hàm không đồng bộ chạy song song, giúp tăng tốc quá trình rất nhiều. Mã giống nhau mất nhiều thời gian hơn nếu nó chạy tuần tự, vì nó không làm gì trong khi chờ mô hình phản ứng. Thông thường nên sử dụng async/await khi chức năng của bạn phải chờ
Ví dụ: để nhận phản hồi từ API hoặc nếu API tải xuống dữ liệu và có thể mất chút thời gian.

Lưu ý sự hiện diện của Semaphore : nó đặt số lượng truy vấn tối đa có thể chạy trong song song. Bạn nên sử dụng Semaphore khi gọi API để tránh lỗi giới hạn tốc độ.

Hãy dùng nó để gọi tới model microsoft/Phi-3-mini-4k-instruct và yêu cầu nó quay lại chủ đề chính của từng bài toán trong bộ dữ liệu Maxwell-Jia/AIME_2024:

```
>>> from datasets import load_dataset
>>> ds = load_dataset("Maxwell-Jia/AIME_2024", split="train")
>>> model = "microsoft/Phi-3-mini-4k-instruct"
>>> prompt = 'What is this text mainly about ? Here is the text:\n\n```\n{Problem}\n```\n\nReply u
>>> async def get_topic(example):
...return await query_model(model, prompt.format(Problem=example['Problem']))
>>> ds = ds.map(get_topic)
>>> ds[0]
{'ID': '2024-II-4',
  'Vấn đề': 'Cho $x,y$ và $z$ là các số thực dương mà...',
  'Solution': 'Denote $\\log_2(x) = a$, $\\log_2(y) = b$, and...,
  'Trả lời': 33,
  'Output': 'The main topic is Logarithms.'}
```

Here, Dataset.map() runs many get_topic function asynchronously so it doesn't have to wait
đối với mỗi phản hồi của một mô hình sẽ mất rất nhiều thời gian để thực hiện tuần tự.

By default, Dataset.map() runs up to one thousand map functions in parallel, so don't forget to
đặt số lượng lệnh gọi API tối đa có thể chạy song song với Semaphore, nếu không thì
mô hình có thể trả về lỗi giới hạn tốc độ hoặc quá tải. Đối với các trường hợp sử dụng nâng cao, bạn có thể t
số lượng truy vấn song song tối đa trong bộ dữ liệu.config.

Xử lý nhiều phần tách

Many datasets have splits that can be processed simultaneously with DatasetDict.map(). For
ví dụ: mã hóa trường câu1 trong tập lệnh và phân chia bài kiểm tra theo:

```
>>> from datasets import load_dataset

# tải tất cả các phần tách
>>> dataset = load_dataset('nyu-mll/glue', 'mrpc')
>>> encoded_dataset = dataset.map(lambda examples: tokenizer(examples["sentence1"]), batched=True)
>>> encoded_dataset["train"][0]
{'sentence1': 'Amrozi accused his brother , whom he called " the witness " , of deliberately disto
'sentence2': 'Chỉ coi anh ta là "nhân chứng " , Amrozi buộc tội anh trai mình cố ý
'nhãn': 1,
'idx': 0,
'input_ids': [101,7277,2180,5303,4806,1117,1711,117,2292, 1119,1270,107,
'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],
'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]
}
```

Sử dụng phân tán

When you use map() in a distributed setting, you should also use torch.distributed.barrier. This đảm bảo quy trình chính thực hiện ánh xạ, trong khi các quy trình khác tải kết quả, do đó tránh được công việc trùng lặp.

Ví dụ sau đây cho thấy cách bạn có thể sử dụng torch.distributed.barrier để đồng bộ hóa quá trình:

```
>>> from datasets import Dataset
>>> import torch.distributed

>>> dataset1 = Dataset.from_dict({"a": [0, 1, 2]})

>>> if training_args.local_rank > 0:
...print("Waiting for main process to perform the mapping")
...torch.distributed.barrier()

>>> dataset2 = dataset1.map(lambda x: {"a": x["a"] + 1})

>>> if training_args.local_rank == 0:
...print("Loading results from main process")
...torch.distributed.barrier()
```

Lô

The batch() method allows you to group samples from the dataset into batches. This is đặc biệt hữu ích khi bạn muốn tạo các lô dữ liệu để đào tạo hoặc đánh giá, đặc biệt là khi làm việc với các mô hình deep learning.

Here's an example of how to use the batch() method:

```
>>> from datasets import load_dataset
>>> dataset = load_dataset("cornell-movie-review-data/rotten_tomatoes", split="train")
>>> batched_dataset = dataset.batch(batch_size=4)
>>> batched_dataset[0]
{'text': ['the rock is destined to be the 21st century\'s new " conan " and that he\'s going to ma
          'Phần tiếp theo được xây dựng công phu và lộng lẫy của bộ ba phim "chúa tể những chiếc nhẫn" l
          'phim tiểu sử hiệu quả nhưng quá tẻ nhạt',
          'nếu thỉnh thoảng bạn thích đi xem phim để giải trí thì wasabi là một nơi tốt để bắt đầu.
 'label': [1, 1, 1, 1]}
```

The batch() method accepts the following parameters:

- batch_size (int): The number of samples in each batch.
- drop_last_batch (bool , defaults to False): Whether to drop the last incomplete batch if kích thước tập dữ liệu không chia hết cho kích thước lô.
- num_proc (int , optional, defaults to None): The number of processes to use for đa xử lý. Nếu Không, không có đa xử lý nào được sử dụng. Điều này có thể tăng tốc đáng kể phân nhóm cho các tập dữ liệu lớn.

Note that Dataset.batch() returns a new Dataset where each item is a batch of multiple mẫu từ tập dữ liệu gốc. Nếu bạn muốn xử lý dữ liệu theo đợt, bạn nên sử dụng batched map() directly, which applies a function to batches but the output dataset is không theo đợt.

Nối

Các bộ dữ liệu riêng biệt có thể được nối nếu chúng có cùng loại cột. Nối datasets with concatenate_datasets():

```
>>> from datasets import concatenate_datasets, load_dataset

>>> stories = load_dataset("ajibawa-2023/General-Stories-Collection", split="train")
>>> stories = stories.remove_columns([col for col in stories.column_names if col != "text"])# on
>>> wiki = load_dataset("wikimedia/wikipedia", "20220301.en", split="train")
>>> wiki = wiki.remove_columns([col for col in wiki.column_names if col != "text"])# only keep t

>>> assert stories.features.type == wiki.features.type
>>> bert_dataset = concatenate_datasets([stories, wiki])
```

You can also concatenate two datasets horizontally by setting axis=1 as long as the datasets
có cùng số hàng:

```
>>> from datasets import Dataset
>>> stories_ids = Dataset.from_dict({"ids": list(range(len(stories)))})
>>> stories_with_ids = concatenate_datasets([stories, stories_ids], axis=1)
```

Xen kẽ

Bạn cũng có thể kết hợp nhiều tập dữ liệu lại với nhau bằng cách lấy các ví dụ xen kẽ từ mỗi tập dữ liệu này s
tạo một tập dữ liệu mới. Điều này được gọi là xen kẽ, được kích hoạt bởi
interleave_datasets() function. Both interleave_datasets() and concatenate_datasets() work
với các đối tượng Dataset và IterableDataset thông thường.
Tham khảo Hướng dẫn luồng để biết ví dụ về cách xen kẽ các đối tượng IterableDataset.

Bạn có thể xác định xác suất lấy mẫu cho từng bộ dữ liệu gốc để chỉ định cách
xen kẽ các tập dữ liệu.
Trong trường hợp này, tập dữ liệu mới được xây dựng bằng cách lấy từng ví dụ một từ ngẫu nhiên
tập dữ liệu cho đến khi một trong các tập dữ liệu hết mẫu.

```
>>> from datasets import Dataset, interleave_datasets
>>> seed = 42
>>> probabilities = [0.3, 0.5, 0.2]
>>> d1 = Dataset.from_dict({"a": [0, 1, 2]})
>>> d2 = Dataset.from_dict({"a": [10, 11, 12, 13]})
>>> d3 = Dataset.from_dict({"a": [20, 21, 22]})
>>> dataset = interleave_datasets([d1, d2, d3], probabilities=probabilities, seed=seed)
>>> dataset["a"]
[10, 11, 20, 12, 0, 21, 13]
```

Bạn cũng có thể chỉ định chiến lược dừng. Chiến lược mặc định, first_exhaused, là một
chiến lược lấy mẫu con, tức là quá trình xây dựng tập dữ liệu bị dừng ngay khi một trong các tập dữ liệu chạy
hết mẫu rồi.
You can specify stopping_strategy=all_exhausted to execute an oversampling strategy. In this
trong trường hợp đó, quá trình xây dựng tập dữ liệu sẽ dừng lại ngay sau khi mọi mẫu trong mỗi tập dữ liệu
được thêm vào ít nhất một lần. Trong thực tế, điều đó có nghĩa là nếu một tập dữ liệu đã hết, nó sẽ trở về trạ
bắt đầu của tập dữ liệu này cho đến khi đạt đến tiêu chí dừng.
Lưu ý rằng nếu không xác định xác suất lấy mẫu thì tập dữ liệu mới sẽ có
mẫu max_length_datasets*nb_dataset.
There is also stopping_strategy=all_exhausted_without_replacement to ensure that every
mẫu được nhìn thấy chính xác một lần.

```
>>> d1 = Dataset.from_dict({"a": [0, 1, 2]})
>>> d2 = Dataset.from_dict({"a": [10, 11, 12, 13]})
>>> d3 = Dataset.from_dict({"a": [20, 21, 22]})
>>> dataset = interleave_datasets([d1, d2, d3], stopping_strategy="all_exhausted")
>>> dataset["a"]
[0, 10, 20, 1, 11, 21, 2, 12, 22, 0, 13, 20]
```

Định dạng

The with_format() function changes the format of a column to be compatible with some
các định dạng dữ liệu phổ biến. Chỉ định đầu ra bạn muốn trong tham số loại. Bạn cũng có thể
choose which the columns you want to format using columns= . Formatting is applied on-the-fly.

For example, create PyTorch tensors by setting type="torch" :

```
>>> dataset = dataset.with_format(type="torch")
```

The set_format() function also changes the format of a column, except it runs in-place:

```
>>> dataset.set_format(type="torch")
```

If you need to reset the dataset to its original format, set the format to None (or use reset_format()):

```
>>> dataset.format
{'type': 'torch', 'format_kwargs': {}, 'columns': [...], 'output_all_columns': False}
>>> dataset = dataset.with_format(None)
>>> dataset.format
{'type': None, 'format_kwargs': {}, 'columns': [...], 'output_all_columns': False}
```

Định dạng tensor

Một số định dạng tensor hoặc mảng được hỗ trợ. Nói chung nên sử dụng những thứ này định dạng thay vì chuyển đổi kết quả đầu ra của tập dữ liệu thành tensor hoặc mảng theo cách thủ công để t sao chép dữ liệu không cần thiết và tăng tốc độ tải dữ liệu.

Dưới đây là danh sách các định dạng tensor hoặc mảng được hỗ trợ:

- NumPy: tên định dạng là "numpy", để biết thêm thông tin, hãy xem Sử dụng Bộ dữ liệu với NumPy
- PyTorch: tên định dạng là "torch", để biết thêm thông tin, hãy xem Sử dụng Bộ dữ liệu với PyTorch
- TensorFlow: tên định dạng là "tensorflow", để biết thêm thông tin, hãy xem Sử dụng Bộ dữ liệu với Dòng chảy căng
- JAX: tên định dạng là "jax", để biết thêm thông tin, hãy xem Sử dụng Bộ dữ liệu với JAX

[!TIP]
Hãy xem hướng dẫn Sử dụng bộ dữ liệu với TensorFlow để biết thêm chi tiết về cách sử dụng hiệu quả tạo tập dữ liệu TensorFlow.

Khi một tập dữ liệu được định dạng theo định dạng tensor hoặc mảng, tất cả dữ liệu sẽ được định dạng dướ or arrays (except unsupported types like strings for example for PyTorch):

```
>>> ds = Dataset.from_dict({"text": ["foo", "bar"], "tokens": [[0, 1, 2], [3, 4, 5]]})
>>> ds = ds.with_format("torch")
>>> ds[0]
{'text': 'foo', 'tokens': tensor([0, 1, 2])}
>>> ds[:2]
{'text': ['foo', 'bar'],
 'tokens': tensor([[0, 1, 2],
                   [3, 4, 5]])}
```

Định dạng bảng

Bạn có thể sử dụng định dạng khung dữ liệu hoặc bảng để tối ưu hóa việc tải dữ liệu và xử lý dữ liệu, vì họ thường cung cấp các hoạt động không sao chép và chuyển đổi được viết bằng ngôn ngữ cấp thấp.

Dưới đây là danh sách các định dạng bảng hoặc khung dữ liệu được hỗ trợ:

- Pandas: tên định dạng là "pandas", để biết thêm thông tin, hãy xem Sử dụng bộ dữ liệu với Pandas
- Cực: tên định dạng là "cực", để biết thêm thông tin, hãy xem Sử dụng bộ dữ liệu với cực
- PyArrow: tên định dạng là "mũi tên", để biết thêm thông tin, hãy xem Sử dụng Bộ dữ liệu với PyArrow

Khi một tập dữ liệu được định dạng theo định dạng khung dữ liệu hoặc bảng, mỗi hàng hoặc lô tập dữ liệu các hàng được định dạng dưới dạng khung dữ liệu hoặc bảng và các cột của tập dữ liệu được định dạng dướ mảng:

```
>>> ds = Dataset.from_dict({"text": ["foo", "bar"], "label": [0, 1]})
>>> ds = ds.with_format("pandas")
>>> ds[:2]
   nhãn văn bản
0foo0
1thanh1
```

Những định dạng này giúp bạn có thể lặp lại dữ liệu nhanh hơn bằng cách tránh các bản sao dữ liệu, đồng th enable faster data processing in map() or filter():

```
>>> ds = ds.map(lambda df: df.assign(upper_text=df.text.str.upper()), batched=True)
>>> ds[:2]
   nhãn văn bản Upper_text
0foo0FOO
1thanh1THANH
```

Chuyển đổi định dạng tùy chỉnh

The with_transform() function applies a custom formatting transform on-the-fly. This function thay thế bất kỳ định dạng nào được chỉ định trước đó. Ví dụ: bạn có thể sử dụng chức năng này để mã hóa và đệm mã thông báo một cách nhanh chóng. Mã thông báo chỉ được áp dụng khi các ví dụ được truy cập:

```
>>> from transformers import AutoTokenizer

>>> tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
>>> def encode(batch):
...return tokenizer(batch["sentence1"], batch["sentence2"], padding="longest", truncation=Tru
>>> dataset = dataset.with_transform(encode)
>>> dataset.format
{'type': 'custom', 'format_kwargs': {'transform': <function __main__.encode(batch)>}, 'columns': [
```

There is also set_transform() which does the same but runs in-place.

You can also use the with_transform() function for custom decoding on Features.

Ví dụ bên dưới sử dụng gói pydub thay thế cho giải mã torchcodec:

```
>>> import numpy as np
>>> from pydub import AudioSegment

>>> audio_dataset_amr = Dataset.from_dict({"audio": ["audio_samples/audio.amr"]})

>>> def decode_audio_with_pydub(batch, sampling_rate=16_000):
...def pydub_decode_file(audio_path):
...sound = AudioSegment.from_file(audio_path)
...if sound.frame_rate != sampling_rate:
...sound = sound.set_frame_rate(sampling_rate)
...channel_sounds = sound.split_to_mono()
...samples = [s.get_array_of_samples() for s in channel_sounds]
...fp_arr = np.array(samples).T.astype(np.float32)
...fp_arr /= np.iinfo(samples[0].typecode).max
...trả về fp_arr
None
...batch["audio"] = [pydub_decode_file(audio_path) for audio_path in batch["audio"]]
...lô hàng trả lại

>>> audio_dataset_amr.set_transform(decode_audio_with_pydub)
```

Cứu

Khi tập dữ liệu của bạn đã sẵn sàng, bạn có thể lưu nó dưới dạng Tập dữ liệu ôm mặt ở định dạng Parquet v
reuse it later with load_dataset().

Lưu tập dữ liệu của bạn bằng cách cung cấp tên của kho lưu trữ tập dữ liệu trên Ôm mặt mà bạn muốn
to save it to to push_to_hub():

```
encoded_dataset.push_to_hub("username/my_dataset")
```

Bạn có thể sử dụng nhiều quy trình để tải nó lên song song. Điều này đặc biệt hữu ích nếu bạn muốn
tăng tốc quá trình:

```
dataset.push_to_hub("username/my_dataset", num_proc=8)
```

Use the load_dataset() function to reload the dataset (in streaming mode or not):

```
từ tập dữ liệu nhập Load_dataset
reloaded_dataset = load_dataset("username/my_dataset", streaming=True)
```

Ngoài ra, bạn có thể lưu cục bộ ở định dạng Mũi tên trên đĩa. So với Parquet, Arrow là không bị nén, giúp tải lại nhanh hơn nhiều, điều này rất tốt cho việc sử dụng cục bộ trên đĩa và bộ nhớ đệm tạm thời. Nhưng vì nó lớn hơn và có ít siêu dữ liệu hơn nên tải lên/tải xuống/truy vấn hơn Parquet và ít phù hợp hơn cho việc lưu trữ lâu dài.

Use the save_to_disk() and load_from_disk() function to reload the dataset from your disk:

```
>>> encoded_dataset.save_to_disk("path/of/my/dataset/directory")
>>> # later
>>> from datasets import load_from_disk
>>> reloaded_dataset = load_from_disk("path/of/my/dataset/directory")
```

Xuất khẩu

Bộ dữ liệu cũng hỗ trợ xuất để bạn có thể làm việc với tập dữ liệu của mình trong các ứng dụng khác. Bảng sau hiển thị các định dạng tệp hiện được hỗ trợ mà bạn có thể xuất sang:

Loại tệp	Phương thức xuất
CSV	Dataset.to_csv()
JSON	Dataset.to_json()
Parquet	Dataset.to_parquet()
SQL	Dataset.to_sql()
Python trong bộ nhớ sự vật	Dataset.to_pandas(), Dataset.to_polars() or Dataset.to_dict()

Ví dụ: xuất tập dữ liệu của bạn sang tệp CSV như thế này:

```
>>> encoded_dataset.to_csv("path/of/my/dataset.csv")
```