

Big data? 🤖 Datasets спешат на помощь!


🔒 Open in Colab 📄 Open Studio Lab

В настоящее время нередко приходится работать с многогигабайтными наборами данных, особенно если вы планируете предварительно обучить трансформер, такой как BERT или GPT-2, с нуля. В этих случаях даже *загрузка* данных может стать проблемой. Например, корпус WebText, используемый для предобучения GPT-2, состоит из более чем 8 миллионов документов и 40 ГБ текста — загрузка этого в оперативную память вашего ноутбука может привести к сердечному приступу!

К счастью, 🤖 Datasets спроектирована так, что позволит избежать таких ограничений. Библиотека избавляет вас от необходимости управлять памятью и рассматривает датасеты как **файлы, отображаемые в память** (memory-mapped files, MMF), также обходит ограничения жестких дисков путем формирования потоков записей из корпуса текстов.

Memory mapping & streaming

Копировать...



Посмотреть на YouTube

В этом разделе мы рассмотрим эти особенности 🤖 Datasets с огромным корпусом объемом 825 ГБ, известным как [Pile] (<https://pile.eleuther.ai>). Давайте начнем!

Что такое the Pile?

The Pile — это корпус текстов на английском языке, созданный [EleutherAI] (<https://www.eleuther.ai>) для обучения крупномасштабных языковых моделей. Он включает в себя широкий спектр наборов данных, включая научные статьи, репозитории кода GitHub и отфильтрованный веб-текст. Учебный корпус доступен в виде [фрагментов по 14 ГБ] (<https://mystic.the-eye.eu/public/AI/pile/>), и вы также можете загрузить несколько [отдельных компонентов] (https://mystic.the-eye.eu/public/AI/pile_preliminary_components/). Начнем с набора данных PubMed Abstracts, который представляет собой свод аннотаций из 15 миллионов биомедицинских публикаций в [PubMed] (<https://pubmed.ncbi.nlm.nih.gov/>). Набор данных находится в [формате JSON Lines] (<https://jsonlines.org>) и сжат с использованием библиотеки `zstandard`, поэтому сначала нам нужно установить библиотеку `zstandard`:

```
!pip install zstandard
```

Затем мы можем загрузить набор данных, используя метод для подгрузки файлов, который мы изучили в [разделе 2](#):

```
from datasets import load_dataset

# Этой займет несколько минут, пока ожидаете - сделайте кофе или чай :)
data_files = "https://mystic.the-eye.eu/public/AI/pile_preliminary_components/PUBMED_title_abstracts_2015"
pubmed_dataset = load_dataset("json", data_files=data_files, split="train")
pubmed_dataset
```

```
Dataset({
  features: ['meta', 'text'],
  num_rows: 15518009
})
```

Мы видим, что в нашем наборе данных 15 518 009 строк и 2 столбца — это очень много!

🔗 По умолчанию 🤖 Datasets распаковывает файлы, необходимые для загрузки набора данных. Если вы хотите сохранить место на жестком диске, вы можете передать `DownloadConfig(delete_extracted=True)` в аргумент `download_config` функции `load_dataset()`. [Дополнительные сведения см. в документации.](#)

Давайте посмотрим на содержимое первого экземпляра:

```
pubmed_dataset[0]
```

```
{'meta': {'pmid': 11409574, 'language': 'eng'},
 'text': 'Epidemiology of hypoxaemia in children with acute lower respiratory infection.\n\nto determine th
```

Отлично, выглядит как аннотация медицинской статьи. Теперь давайте посмотрим объем памяти, который мы использовали при загрузке данных:

Магия отображения в память

Простой способ измерить использование памяти в Python — использовать библиотеку `psutil`, которую можно установить с помощью `pip` следующим образом:

```
!pip install psutil
```

Она предоставляет класс `Process`, который позволяет нам проверить использование памяти текущим процессом следующим образом:

```
import psutil

# Process.memory_info возвращает объем в байтах, мы пересчитаем в мегабайты
print(f"RAM used: {psutil.Process().memory_info().rss / (1024 * 1024):.2f} MB")
```

```
RAM used: 5678.33 MB
```

Здесь атрибут `rss` относится к *резидентному размеру набора*, который представляет собой долю памяти, которую процесс занимает в ОЗУ. Это измерение также включает память, используемую интерпретатором Python и загруженными нами библиотеками, поэтому фактический объем памяти, используемый для загрузки набора данных, немного меньше. Для сравнения давайте посмотрим, насколько велик набор данных на диске, используя атрибут `dataset_size`. Поскольку результат, как и раньше, выражается в байтах, нам нужно вручную преобразовать его в гигабайты:

```
print(f"Number of files in dataset : {pubmed_dataset.dataset_size}")
size_gb = pubmed_dataset.dataset_size / (1024**3)
print(f"Dataset size (cache file) : {size_gb:.2f} GB")
```

```
Number of files in dataset : 26979437051
Dataset size (cache file) : 19.54 GB
```

Приятно — несмотря на то, что он весит почти 20 ГБ, мы можем загрузить и получить доступ к набору данных с гораздо меньшим объемом оперативной памяти!

🔗 **Попробуйте!** Выберите один из [компонентов](#) из Pile, который больше, чем оперативная память вашего ноутбука или настольного компьютера, загрузите его с 🤖 Datasets и измерьте объем используемой оперативной памяти. Обратите внимание, что для получения точных измерений вам потребуются сделать это в новом процессе. Вы можете найти распакованные размеры каждого компонента в Таблице 1 [документации Pile] (<https://arxiv.org/abs/2101.00027>).

Если вы знакомы с Pandas, этот результат может стать неожиданностью из-за знаменитого [эмпирического правила] Уэса Кинни (<https://wesmckinney.com/blog/apache-arrow-pandas-internals/>), согласно которому вам обычно требуется 5 до 10 раз больше оперативной памяти, чем размер вашего набора данных. Так как же 🤖 Datasets решают эту проблему управления памятью? 🤖 Datasets рассматривают каждый набор данных как [файл с отображением в память] (https://en.wikipedia.org/wiki/Memory_mapped_file), который обеспечивает сопоставление между оперативной памятью и хранилищем файловой системы, что позволяет библиотеке получать доступ к элементам и работать с ними без необходимости полной загрузки его в память.

Memory-mapped файлы также могут совместно использоваться несколькими процессами, что позволяет распараллеливать такие методы, как `Dataset.map()`, без необходимости перемещать или копировать набор данных. Под капотом все эти возможности реализованы в формате [Apache Arrow](#) и [ryarrow] (<https://arrow.apache.org/docs/python/index.html>), которые делают загрузку и обработку данных молниеносной. (Для получения более подробной информации об Apache Arrow и сравнении с Pandas ознакомьтесь с [публикацией в блоге Денна Симикья] (<https://towardsdatascience.com/apache-arrow-read-dataframe-with-zero-memory-69634092b1a>). Чтобы увидеть это в действии давайте проведем небольшой тест скорости, перебирая все элементы в наборе данных PubMed Abstracts:

```
import timeit

code_snippet = """batch_size = 1000

for idx in range(0, len(pubmed_dataset), batch_size):
    _ = pubmed_dataset[idx:idx + batch_size]
"""

time = timeit.timeit(stmt=code_snippet, number=1, globals=globals())
print(
    f'Iterated over {len(pubmed_dataset)} examples (about {size_gb:.1f} GB) in "
    f"{time:.1f}s, i.e. {size_gb/time:.3f} GB/s"
)
```

```
'Iterated over 15518009 examples (about 19.5 GB) in 64.2s, i.e. 0.304 GB/s'
```

Здесь мы использовали модуль `timeit` Python для измерения времени выполнения `code_snippet`. Обычно вы сможете перебирать набор данных со скоростью от нескольких десятых долей ГБ/с до нескольких ГБ/с. Это прекрасно работает для подавляющего большинства приложений, но иногда вам придется работать с набором данных, который слишком велик даже для хранения на жестком диске вашего ноутбука. Например, если бы мы попытались загрузить весь Pile, нам потребовалось бы 825 ГБ свободного места на диске! Чтобы справиться с такими случаями 🤖 Datasets предоставляют функцию потоковой передачи, которая позволяет нам загружать и получать доступ к элементам на лету, без необходимости загружать весь набор данных. Давайте посмотрим, как это работает.

💡 В Jupyter notebooks вы также можете измерить время исполнения ячейки с использованием [@timeit magic function](#).

Потоковая передача датасета

Чтобы включить потоковую передачу набора данных, вам просто нужно передать аргумент `streaming=True` в функцию `load_dataset()`. Например, давайте снова загрузим набор данных PubMed Abstracts, но в потоковом режиме:

```
pubmed_dataset_streamed = load_dataset(
    "json", data_files=data_files, split="train", streaming=True
)
```

Вместо знакомого `Dataset`, с которым мы уже встречались в других местах этой главы, объект, возвращаемый с `streaming=True`, является `IterableDataset`. Как следует из названия, чтобы получить доступ к элементам `IterableDataset`, нам нужно заполнить итерацию по нему. Мы можем получить доступ к первому элементу нашего набора потоковых данных следующим образом:

```
next(iter(pubmed_dataset_streamed))
```

```
{'meta': {'pmid': 11409574, 'language': 'eng'},
 'text': 'Epidemiology of hypoxaemia in children with acute lower respiratory infection.\n\nto determine th
```

Элементы из потокового набора данных можно обрабатывать на лету с помощью `IterableDataset.map()`, что полезно во время обучения, если вам нужно токенизировать входные данные. Процесс точно такой же, как тот, который мы использовали для токенизации нашего набора данных в [Главе 3](#), с той лишь разницей, что выходные данные возвращаются один за другим:

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
tokenized_dataset = pubmed_dataset_streamed.map(lambda x: tokenizer(x["text"]))
next(iter(tokenized_dataset))
```

```
{'input_ids': [101, 4958, 5178, 4328, 6779, ...], 'attention_mask': [1, 1, 1, 1, 1, ...]}
```

💡 Чтобы ускорить токенизацию с потоковой передачей, вы можете передать `batched=True`, как мы делали в [последнем разделе](#). Он будет обрабатывать примеры батчами; размер батча по умолчанию составляет 1000 и может быть указан в аргументе `batch_size`.

Вы также можете перемешать потоковые наборы данных, используя `IterableDataset.shuffle()`, но в отличие от `Dataset.shuffle()`, это только перемешивает элементы в предопределенном `buffer_size`:

```
shuffled_dataset = pubmed_dataset_streamed.shuffle(buffer_size=10_000, seed=42)
next(iter(shuffled_dataset))
```

```
{'meta': {'pmid': 11410799, 'language': 'eng'},
 'text': 'Randomized study of dose or schedule modification of granulocyte colony-stimulating factor in p
```

В этом примере мы выбрали случайный пример из первых 10 000 примеров в буфере. После обращения к примеру его место в буфере заполняется следующим примером в корпусе (т. е. 10 001-м примером в приведенном выше случае). Вы также можете выбирать элементы из потокового набора данных, используя функции `IterableDataset.take()` и `IterableDataset.skip()`, которые действуют аналогично `Dataset.select()`. Например, чтобы выбрать первые 5 примеров в наборе данных PubMed Abstracts, мы можем сделать следующее:

```
dataset_head = pubmed_dataset_streamed.take(5)
list(dataset_head)
```

```
[{'meta': {'pmid': 11409574, 'language': 'eng'},
 'text': 'Epidemiology of hypoxaemia in children with acute lower respiratory infection ...'},
 {'meta': {'pmid': 11409575, 'language': 'eng'},
 'text': 'Clinical signs of hypoxaemia in children with acute lower respiratory infection: indicators of'},
 {'meta': {'pmid': 11409576, 'language': 'eng'},
 'text': 'Hypoxaemia in children with severe pneumonia in Papua New Guinea ...'},
 {'meta': {'pmid': 11409577, 'language': 'eng'},
 'text': 'Oxygen concentrators and cylinders ...'},
 {'meta': {'pmid': 11409578, 'language': 'eng'},
 'text': 'Oxygen supply in rural africa: a personal experience ...'}]
```

Точно так же вы можете использовать функцию `IterableDataset.skip()` для создания обучающих и проверочных сплитов из перемешанного набора данных следующим образом:

```
# Пропустить первые 1000 объектов и включить остальные в обучающую выборку
train_dataset = shuffled_dataset.skip(1000)
# Взять первые 1000 объектов в валидационную выборку
validation_dataset = shuffled_dataset.take(1000)
```

Давайте завершим наше исследование потоковой передачи наборов данных общим приложением: объединение нескольких наборов данных вместе для создания единого корпуса. 🤖 Datasets предоставляют функцию `interleave_datasets()`, которая преобразует список объектов `IterableDataset` в один `IterableDataset`, где элементы нового набора данных получают путем чередования исходных примеров. Эта функция особенно полезна, когда вы пытаетесь объединить большие наборы данных, поэтому в качестве примера давайте воспроизведем компонент `FreeLaw` из Pile, который представляет собой набор данных юридических заключений судов США объемом 51 ГБ:

```
law_dataset_streamed = load_dataset(
    "json",
    data_files="https://mystic.the-eye.eu/public/AI/pile_preliminary_components/FreeLaw_Opinions.jsonl.zst",
    split="train",
    streaming=True,
)
next(iter(law_dataset_streamed))
```

```
{'meta': {'case_ID': '110921.json',
 'case_jurisdiction': 'scotus.tar.gz',
 'date_created': '2010-04-28T17:12:49Z'},
 'text': '\n461 U.S. 238 (1983)\nOLIM ET AL.\nv.\nWAKINEKONA\nNo. 81-1581.\nSupreme Court of United State
```

Этот набор данных достаточно велик, чтобы нагружать оперативную память большинства ноутбуков, но мы смогли загрузить его и получить к нему доступ! Давайте теперь объединим примеры из наборов данных `FreeLaw` и `PubMed Abstracts` с функцией `interleave_datasets()`:

```
from itertools import islice
from datasets import interleave_datasets

combined_dataset = interleave_datasets([pubmed_dataset_streamed, law_dataset_streamed])
list(islice(combined_dataset, 2))
```

```
[{'meta': {'pmid': 11409574, 'language': 'eng'},
 'text': 'Epidemiology of hypoxaemia in children with acute lower respiratory infection ...'},
 {'meta': {'case_ID': '110921.json',
 'case_jurisdiction': 'scotus.tar.gz',
 'date_created': '2010-04-28T17:12:49Z'},
 'text': '\n461 U.S. 238 (1983)\nOLIM ET AL.\nv.\nWAKINEKONA\nNo. 81-1581.\nSupreme Court of United Sta
```

Здесь мы использовали функцию `islice()` из модуля `itertools` Python, чтобы выбрать первые два объекта из объединенного набора данных, и мы видим, что они соответствуют первым примерам из каждого из двух исходных наборов данных.

Наконец, если вы хотите получить в потоковом режиме весь Pile целиком (825 ГБ), вы можете получить все подготовленные файлы следующим образом:

```
base_url = "https://mystic.the-eye.eu/public/AI/pile/"
data_files = {
    "train": [base_url + "train/" + f"{idx:02d}.jsonl.zst" for idx in range(30)],
    "validation": base_url + "val.jsonl.zst",
    "test": base_url + "test.jsonl.zst",
}
```

```
pile_dataset = load_dataset("json", data_files=data_files, streaming=True)
next(iter(pile_dataset["train"]))
```

```
{'meta': {'pile_set_name': 'Pile-CC'},
 'text': 'It is done, and submitted. You can play “Survival of the Tastiest” on Android, and on the web. .
```

🔗 **Попробуйте!** Используйте один из больших корпусов Common Crawl, например `mc4` или `oscar`] (<https://huggingface.co/datasets/oscar>) для создания потокового многоязычного набора данных, который представляет пропорции разговорных языков в стране по вашему выбору. Например, в Швейцарии есть четыре национальных языка: немецкий, французский, итальянский и рето-романский, поэтому вы можете попробовать создать швейцарский корпус, выбрав подмножества Оскаров в соответствии с их разговорной пропорцией.

Теперь у вас есть все инструменты, необходимые для загрузки и обработки наборов данных всех форм и размеров, но, если только вам не повезет, в вашем путешествии по НЛП наступит момент, когда вам придется фактически создать собственный набор данных для решения проблемы. Это тема следующего раздела!

➔ [Update](#) on GitHub