Трансформеры: на что они способны?

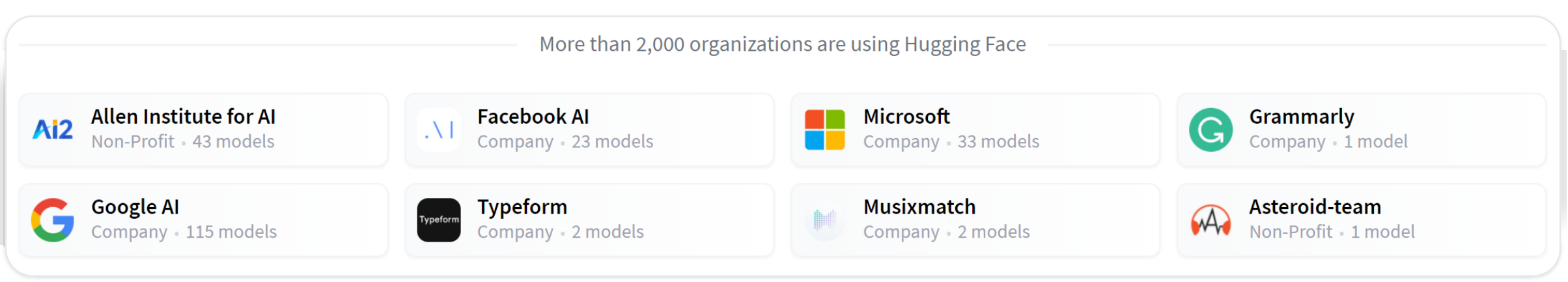
Ask a question | Open in Colab

В этом разделе мы посмотрим, на что способны трансформеры и первый раз применим функцию из библиотеки 🤗 Transformers: `pipeline()`.

> 👀 Видите кнопку *Open in Colab* справа сверху? Нажмите на нее, чтобы открыть блокнот в Google Colab с примерами кода этого раздела. Эта кнопка будет во всех разделах, которые содержат примеры кода.
> Если вы хотите запускать ноутбуки локально, то обратите внимание на раздел setup.

Трансформеры повсюду!

Трансформеры используются для решения всех видов задач NLP, перечисленных в предыдущем разделе. Библиотека Transformers используется некоторыми компаниями и организациями, которые делятся своими моделями с сообществом Hugging Face:



Библиотека 🤗 Transformers предоставляет различную функциональность для создания и использования этих моделей. Model Hub содержит тысячи предобученных моделей, которые может скачать и использовать любой. Вы также можете загружать свои модели на Model Hub!

> ⚠️ Hugging Face Hub не ограничивается только моделями. Любой человек может поделиться своими моделями или датасетами! Для этого нужно создать учетную запись: Create a huggingface.co

Прежде чем погрузиться в механизм работы трансформеров, давайте взглянем на несколько примеров того, как они могут быть использованы для решения задач NLP.

Работа с пайплайнами

Самый базовый объект библиотеки 🤗 Transformers - `pipeline()`. Он соединяет в себе необходимые шаги по предобработке и постобработке данных и позволяет передавать модели на вход текст, а на выходе получать читаемый ответ:

```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
classifier("I've been waiting for a HuggingFace course my whole life.")
```

```
[{'label': 'POSITIVE', 'score': 0.9598047137260437}]
```

Мы можем передать на вход сразу несколько предложений!

```
classifier(
    ["I've been waiting for a HuggingFace course my whole life.", "I hate this so much!"]
)
```

```
[{'label': 'POSITIVE', 'score': 0.9598047137260437},
 {'label': 'NEGATIVE', 'score': 0.9994558095932007}]
```

По умолчанию этот пайплайн выбирает специальную модель, которая была предобучена для оценки тональности предложений на английском языке. Модель загружается и кэшируется когда вы создадаете объект `classifier`. Если вы перезапустите команду, будет использована кэшированная модель, т.е. загрузки новой модели не произойдет.

В процессе обработки пайплайном текста, который вы ему передали, есть три главных шага:

1. Текст предобрабатывается в формат, который понятен модели.
2. Предобработанный текст передается в модель.
3. Результаты модели проходят через постобработку и вы получаете читаемый ответ.

Некоторые из доступных пайплайнов are:

- `feature-extraction` (превращение текста в векторы)
- `fill-mask`
- `ner` (распознавание именованных сущностей)
- `question-answering`
- `sentiment-analysis`
- `summarization`
- `text-generation`
- `translation`
- `zero-shot-classification`

Давайте взглянем на некоторые из них!

Zero-shot classification

Мы начнем с более сложной задачи, где нам нужно классифицировать тексты, которые не были размечены (для них нет ответов - позитивный или негативный, агрессивный или нейтральный и тд). Это распространенный сценарий в реальных проектах, потому что разметка текста обычно занимает много времени и требует знаний в предметной области. Для этого варианта использования очень мощным является пайплайн `zero-shot-classification`: он позволяет указать, какие метки использовать для классификации, поэтому вам не нужно полагаться на метки предварительно обученной модели. Вы уже видели, как модель может классифицировать предложение как позитивное или негативное, используя эти две метки, но она также может классифицировать текст, используя любой другой набор меток, который вам подходит.

```
from transformers import pipeline

classifier = pipeline("zero-shot-classification")
classifier(
    "This is a course about the Transformers library",
    candidate_labels=["education", "politics", "business"],
)
```

```
{'sequence': 'This is a course about the Transformers library',
 'labels': ['education', 'business', 'politics'],
 'scores': [0.8445963859558105, 0.111976258456707, 0.043427448719739914]}
```

Этот пайплайн называется *zero-shot* потому, что вам нет необходимости дообучать модель для использования. Она может вернуть вам оценки вероятности для любого списка меток, который вы хотите!

> ✏️ **Попробуйте** Поэкспериментируйте с собственными предложениями и метками, и посмотрите как ведет себя модель!

Генерация текста

Теперь давайте взглянем на пайплайн генерации текста (англ. text generation). Главная идея заключается в следующем: вы передаете на вход модели небольшой фрагмент текста, а модель будет продолжать его. Это похоже на предсказание следующего слова в клавиатурах различных смартфонов. Генерация текста содержит в себе элемент случайности, поэтому ваш результат может отличаться от того, который приведен ниже в примере.

```
from transformers import pipeline

generator = pipeline("text-generation")
generator("In this course, we will teach you how to")
```

```
[{'generated_text': 'In this course, we will teach you how to understand and use '
                    'data flow and data interchange when handling user data. We '
                    'will be working with one or more of the most commonly used '
                    'data flows — data flows of various types, as seen by the '
                    'HTTP'}]
```

Вы можете указать, сколько разных «ответов» сгенерирует модель, задав параметр `num_return_sequences`. Чтобы изменить длину ответной последовательности, нужно передать значение в аргумент `max_length`.

> ✏️ **Попробуйте!** Измените `num_return_sequences` и `max_length` так, чтобы были сгенерированы два ответа каждый из которых будет состоять из 15 слов.

Использование произвольной модели из Hub в пайплайне

Предыдущие примеры использовали модель по умолчанию для решения конкретной задачи, но у вас есть возможность выбрать произвольную модель из Hub и передать ее в пайплайн для конкретной задачи. Например, для генерации текста. Перейдите по ссылке Model Hub и кликните на соответствующий тег слева, чтобы получить список доступных для этой задачи моделей. Вы должны увидеть страницу, подобную этой.

Давайте попробуем модель distilgpt2! Тут показано, как загрузить ее в такой же пайплайн, как в примерах выше:

```
from transformers import pipeline

generator = pipeline("text-generation", model="distilgpt2")
generator(
    "In this course, we will teach you how to",
    max_length=30,
    num_return_sequences=2,
)
```

```
[{'generated_text': 'In this course, we will teach you how to manipulate the world and '
                    'move your mental and physical capabilities to your advantage.'},
 {'generated_text': 'In this course, we will teach you how to become an expert and '
                    'practice realtime, and with a hands on experience on both real '
                    'time and real'}]
```

Вы можете уточнить, для какого языка вам нужна модель, щелкнув на языковые теги, и выбрать ту, которая будет генерировать текст на другом языке. На Model Hub даже есть предобученные модели для многоязычных задач.

Как только вы выберете модель, вы увидите, что есть виджет, позволяющий вам попробовать ее прямо на сайте. Таким образом, вы можете быстро протестировать возможности модели перед ее загрузкой.

> ✏️ **Попробуйте!** Найдите модель для другого языка и используйте виджет для проверки!

The Inference API

Все модели могут быть протестированы прямо на сайте с использованием inference API, доступного по адресу https://huggingface.co/. Вы можете попробовать применить модель, вводя различный текст и сразу же получая результат.

Inference API также представляется как платный продукт, что пригодится для интегрирования моделей в свои рабочие процессы. Подробнее можно узнать на странице с ценами.

Заполнение пропусков

Следующая задача, на которую мы обратим внимание, связана с заполнением пропусков в тексте (англ. mask filling). Идея очень проста, мы продемонстрируем ее на простом тексте:

```
from transformers import pipeline

unmasker = pipeline("fill-mask")
unmasker("This course will teach you all about <mask> models.", top_k=2)
```

```
[{'sequence': 'This course will teach you all about mathematical models.',
  'score': 0.19619831442832947,
  'token': 30412,
  'token_str': ' mathematical'},
 {'sequence': 'This course will teach you all about computational models.',
  'score': 0.04052725434303284,
  'token': 38163,
  'token_str': ' computational'}]
```

Аргумент `top_k` указывает, сколько вариантов для пропущенного слова будет отображено. Обратите внимание, что модель заполнит пропуск на месте слова `<mask>`, которое часто интерпретируют как *mask token (токен-маска)*. Другие модели могут использовать другие токены для обозначения пропуска, всегда лучше проверять это. Один из способов сделать это - обратить внимание на виджет для соответствующей модели.

> ✏️ **Попробуйте!** Найдите в поиске модель `bert-based-cased` и обратите внимание на его токен-маску в виджете. Что эта модель предскажет, если применить ее в предыдущем примере?

Распознавание именованных сущностей (NER)

Распознавание именованных сущностей (англ. named entity recognition) - это задача, в которой модели необходимо найти части текста, соответствующие некоторым сущностям, например: персонам, местам, организациям. Давайте посмотрим на пример:

```
from transformers import pipeline

ner = pipeline("ner", grouped_entities=True)
ner("My name is Sylvain and I work at Hugging Face in Brooklyn.")
```

```
[{'entity_group': 'PER', 'score': 0.99816, 'word': 'Sylvain', 'start': 11, 'end': 18},
 {'entity_group': 'ORG', 'score': 0.97960, 'word': 'Hugging Face', 'start': 33, 'end': 45},
 {'entity_group': 'LOC', 'score': 0.99321, 'word': 'Brooklyn', 'start': 49, 'end': 57}
]
```

В этом примере модель корректно обозначила Sylvain как персону (PER), Hugging Face как организацию (ORG) и Brooklyn как локацию (LOC).

Мы передали в пайплайн аргумент `grouped_entities=True` для того, чтобы модель сгруппировала части предложения, соответствующие одной сущности: в данном случае модель объединила "Hugging" и "Face" несмотря на то, что название организации состоит из двух слов. На самом деле, как мы увидим в следующей главе, препроцессинг делит даже отдельные слова на несколько частей. Например, `Sylvain` будет разделено на 4 части: `S`, `##yl`, `##va`, and `##in`. На этапе постпроцессинга пайплайн успешно объединит эти части.

> ✏️ **Попробуйте!** Найдите на Model Hub модель, позволяющую решать задачу определения частей речи в предложении (part of speech tagging, POS). Что модель предскажет для предложения из примера выше?

Вопросно-ответные системы

Пайплайн `question-answering` позволяет сгенерировать ответ на вопрос по данному контексту:

```
from transformers import pipeline

question_answerer = pipeline("question-answering")
question_answerer(
    question="Where do I work?",
    context="My name is Sylvain and I work at Hugging Face in Brooklyn",
)
```

```
{'score': 0.6385916471481323, 'start': 33, 'end': 45, 'answer': 'Hugging Face'}
```

Обратите внимание, что пайплайн извлекает информацию для ответа из переданного ему контекста

Автоматическое реферирование (саммаризация)

Автоматическое реферирование (англ. summarization) - задача, в которой необходимо сократить объем текста, но при этом сохранить все важные аспекты (или большинство из них) изначального текста. Вот пример:

```
from transformers import pipeline

summarizer = pipeline("summarization")
summarizer(
    """
    America has changed dramatically during recent years. Not only has the number of
    graduates in traditional engineering disciplines such as mechanical, civil,
    electrical, chemical, and aeronautical engineering declined, but in most of
    the premier American universities engineering curricula now concentrate on
    and encourage largely the study of engineering science. As a result, there
    are declining offerings in engineering subjects dealing with infrastructure,
    the environment, and related issues, and greater concentration on high
    technology subjects, largely supporting increasingly complex scientific
    developments. While the latter is important, it should not be at the expense
    of more traditional engineering.

    Rapidly developing economies such as China and India, as well as other
    industrial countries in Europe and Asia, continue to encourage and advance
    the teaching of engineering. Both China and India, respectively, graduate
    six and eight times as many traditional engineers as does the United States.
    Other industrial countries at minimum maintain their output, while America
    suffers an increasingly serious decline in the number of engineering graduates
    and a lack of well-educated engineers.
"""
)
```

```
[{'summary_text': ' America has changed dramatically during recent years . The '
                  'number of engineering graduates in the U.S. has declined in '
                  'traditional engineering disciplines such as mechanical, civil '
                  ', electrical, chemical, and aeronautical engineering . Rapidly '
                  'developing economies such as China and India, as well as other '
                  'industrial countries in Europe and Asia, continue to encourage '
                  'and advance engineering .'}]
```

Так же, как и в задаче генерации текста, вы можете указать максимальную длину `max_length` или минимальную длину `min_length` результата.

Перевод

Для перевода вы можете использовать модель по умолчанию просто указав пару языков, между которыми будет осуществляться перевод (например, `"translation_en_to_fr"`). Однако самый простой способ - попробовать использовать Model Hub. Здесь мы попробуем перевести текст с французского на английский язык:

```
from transformers import pipeline

translator = pipeline("translation", model="Helsinki-NLP/opus-mt-fr-en")
translator("Ce cours est produit par Hugging Face.")
```

```
[{'translation_text': 'This course is produced by Hugging Face.'}]
```

Так же, как и в задачах генерации и автоматического реферирования текста, вы можете указать максимальную длину `max_length` или минимальную длину `min_length` результата.

> ✏️ **Попробуйте!** Найдите модель, которая переведет предложение из примера выше на другие языки

Показанные пайплайны в основном носят демонстрационный характер, потому что настроены на решение конкретных задач. В следующей главе вы узнаете, как изменить поведение функции `pipeline()`.