

Przetwarzanie języka naturalnego

Wstęp

Obecnie najpopularniejsze model służące do przetwarzania języka naturalnego wykorzystują architekturę transformacyjną. Istnieje kilka bibliotek, implementujących tę architekturę, ale w kontekście NLP najczęściej wykorzystuje się [Huggingface transformers](#).

Biblioteka ta poza samym [kodem źródłowym](#), zawiera szereg innych elementów. Do najważniejszych z nich należą:

- [modele](#) - olbrzymia i ciągle rosnąca liczba gotowych modeli, których możemy użyć do rozwiązywania wielu problemów z dziedziny NLP (ale również w zakresie rozpoznawania mowy, czy przetwarzania obrazu),
- [zbiory danych](#) - bardzo duży katalog przydatnych zbiorów danych, które możemy w prosty sposób wykorzystać do trenowania własnych modeli NLP (oraz innych modeli).

Weryfikacja dostępności GPU

Trening modeli NLP wymaga dostępu do akceleratorów sprzętowych, przyspieszających uczenie sieci neuronowych. Jeśli nasz komputer nie jest wyposażony w GPU, to możemy skorzystać ze środowiska Google Colab.

 [Open in Colab](#)

W tym środowisku możemy wybrać akcelerator spośród GPU i TPU.

Sprawdźmy, czy mamy dostęp do środowiska wyposażonego w akcelerator NVIDIA:

```
!nvidia-smi
```

```

Sun Dec 15 09:31:30 2024
+-----+
| NVIDIA-SMI 535.104.05                Driver Version: 535.104.05   CUDA Version: 12.2   |
+-----+-----+-----+-----+-----+-----+
| GPU  Name      Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |      Memory-Usage | GPU-Util  Compute M. |
|=====+=====+=====+=====+=====+=====+
|    0  Tesla T4           Off | 00000000:00:04:0  Off |             0%      0 |
| N/A   37C    P8             9W / 70W |  0MiB / 15360MiB |      0%   Default   |
+-----+-----+-----+-----+-----+-----+
|
| Processes:
|  GPU   GI    CI          PID    Type    Process name                  GPU Memory
|   ID   ID     ID              |                   |           Usage         |
|=====+=====+=====+=====+=====+=====+
|  No running processes found
+-----+

```

Jeśli akcelerator jest niedostępny (polecenie skończyło się błędem), to zmieniamy środowisko wykonawcze wybierając z menu "Środowisko wykonawcze" -> "Zmień typ środowiska wykonawczego" -> GPU.

Podpięcie dysku Google (opcjonalne)

Kolejnym elementem przygotowań, który jest opcjonalny, jest dołączenie własnego dysku Google Drive do środowiska Colab. Dzięki temu możliwe jest zapisywanie wytrenowanych modeli, w trakcie procesu treningu, na "zewnętrznym" dysku. Jeśli Google Colab doprowadzi do przerwania procesu treningu, to mimo wszystko pliki, które udało się zapisać w trakcie treningu nie przepadną. Możliwe będzie wznowienie treningu już na częściowo wytrenowanym modelu.

W tym celu montujemy dysk Google w Colabie. Wymaga to autoryzacji narzędzia Colab w Google Drive.

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

 Mounted at /content/gdrive

Po podmontowaniu dysku mamy dostęp do całej zawartości Google Drive. Wskazując miejsce zapisywania danych w trakcie treningu należy wskazać ścieżkę zaczynającą się od [/content/gdrive](#), ale należy wskazać jakiś podkatalog w ramach naszej przestrzeni dyskowej. Pełna ścieżka może mieć postać [/content/gdrive/MyDrive/output](#). Przed uruchomieniem treningu warto sprawdzić, czy dane zapisują się na dysku.

✓ Instalacja bibliotek Pythona

Podobnie jak w poprzednich laboratoriach optymalnym sposobem instalacji bibliotek jest wykorzystanie narzędzia Poetry, które ma ustalone wersje bibliotek w pliku `poetry.lock`. Biblioteki te zostały zmodyfikowane względem wcześniejszych laboratoriów, dlatego ponownie powinniśmy je zainstalować.

```
!poetry install --no-root
```

```
→ /bin/bash: line 1: poetry: command not found
```

Mając zainstalowane niezbędne biblioteki, możemy skorzystać z wszystkich modeli i zbiorów danych zarejestrowanych w katalogu.

Typowym sposobem użycia dostępnych modeli jest:

- *wykorzystanie gotowego modelu*, który realizuje określone zadanie, np. [analizę sentymentu w języku angielskim](#) - model tego rodzaju nie musi być trenowany, wystarczy go uruchomić aby uzyskać wynik klasyfikacji (można to zobaczyć w demo pod wskazanym linkiem),
- *wykorzystanie modelu bazowego*, który jest dotrenowywany do określonego zadania; przykładem takiego modelu jest [HerBERT base](#), który uczony był jako maskowany model języka. Żeby wykorzystać go do konkretnego zadania, musimy wybrać dla niego "głowę klasyfikacyjną" oraz dotrenować na własnym zbiorze danych.

Modele tego rodzaju różnią się od siebie, można je załadować za pomocą wspólnego interfejsu, ale najlepiej jest wykorzystać jedną ze specjalizowanych klas, dostosowanych do zadania, które chcemy zrealizować. Zaczniemy od załadowania modelu BERT base - jednego z najbardziej popularnych modeli, dla języka angielskiego. Za jego pomocą będziemy odgadywać brakujące wyrazy w tekście. Wykorzystamy do tego wywołanie `AutoModelForMaskedLM`.

```
from transformers import AutoModelForMaskedLM, AutoTokenizer
```

```
model = AutoModelForMaskedLM.from_pretrained("bert-base-cased")
```

```
→ /usr/local/lib/python3.10/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 :
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(
config.json: 100% 570/570 [00:00<00:00, 33.9kB/s]
model.safetensors: 100% 436M/436M [00:01<00:00, 229MB/s]
BertForMaskedLM has generative capabilities, as `prepare_inputs_for_generation` is explicitly overwritten. However, it doesn't direc
- If you're using `trust_remote_code=True`, you can get rid of this warning by loading the model with an auto class. See https://
- If you are the owner of the model architecture code, please modify your model class such that it inherits from `GenerationMixin`
- If you are not the owner of the model architecture class, please contact the model code owner to update it.
Some weights of the model checkpoint at bert-base-cased were not used when initializing BertForMaskedLM: ['bert.pooler.dense.bias',
- This IS expected if you are initializing BertForMaskedLM from the checkpoint of a model trained on another task or with another ar
- This IS NOT expected if you are initializing BertForMaskedLM from the checkpoint of a model that you expect to be exactly identica
```

załadowany model jest modulem PyTorch. Możemy zatem korzystać z API tej biblioteki. Możemy np. sprawdzić ile parametrów ma model BERT base:

```
count = sum(p.numel() for p in model.parameters() if p.requires_grad)
```

```
'{:,}'.format(count).replace(',', ' ')
```

```
→ '108 348 881'
```

Widzimy zatem, że nasz model jest bardzo duży - zawiera ponad 100 milionów parametrów, a jest to tzw. model bazowy. Modele obecnie wykorzystywane mają jeszcze więcej parametrów - duże modele językowe, takie jak ChatGPT posiadają więcej niż 100 miliardów parametrów.

Możemy również podejrzeć samą strukturę modelu.

```
model
```

```
→ BertForMaskedLM(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(28996, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
```

```

(encoder): BertEncoder(
  (layer): ModuleList(
    (0-11): 12 x BertLayer(
      (attention): BertAttention(
        (self): BertSdpaSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
        (intermediate_act_fn): GELUActivation()
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
  )
)
)
)
(cls): BertOnlyMLMHead(
  (predictions): BertLMPredictionHead(
    (transform): BertPredictionHeadTransform(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (transform_act_fn): GELUActivation()
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    )
    (decoder): Linear(in_features=768, out_features=28996, bias=True)
  )
)
)

```

Jeśli dysponujemy akceleratorem (GPU lub inny), to pamiętajmy żeby przenieść model na ten akcelerator, np.

```

# Jeśli chcesz użyć akceleratora wpisz "cuda:0" lub nazwę odpowiedniego akceleratora.
# Żeby kod poniżej działał, ale obliczenia były wykonywane na CPU, wpisz "cpu"
#device = "cpu"
device = "cuda:0"
model.to(device)
print("")

```



✓ Tokenizacja tekstu

Łaadowanie samego modelu nie jest jednak wystarczające, żeby zacząć go wykorzystywać. Musimy mieć mechanizm zamiany tekstu (łańcucha znaków), na ciąg tokenów, należących do określonego słownika. W trakcie treningu modelu, słownik ten jest określany (wybierany w sposób algorytmiczny) przed właściwym treningiem sieci neuronowej. Choć możliwe jest jego późniejsze rozszerzenie (douczenie na danych treningowych, pozwala również uzyskać reprezentację brakujących tokenów), to zwykle wykorzystuje się słownik w postaci, która została określona przed treningiem sieci neuronowej. Dlatego tak istotne jest wskazanie właściwego słownika dla tokenizera dokonującego podziału tekstu.

Biblioteka posiada klasę `AutoTokenizer`, która akceptuje nazwę modelu, co pozwala automatycznie załadować słownik korespondujący z wybranym modelem sieci neuronowej. Trzeba jednak pamiętać, że jeśli używamy 2 modeli, to każdy z nich najpewniej będzie miał inny słownik, a co za tym idzie muszą one mieć własne instancje klasy `Tokenizer`.

```

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
tokenizer

```

```

tokenizer_config.json: 100% 49.0/49.0 [00:00<00:00, 2.47kB/s]

vocab.txt: 100% 213k/213k [00:00<00:00, 435kB/s]

tokenizer.json: 100% 436k/436k [00:00<00:00, 951kB/s]

BertTokenizerFast(name_or_path='bert-base-cased', vocab_size=28996, model_max_length=512, is_fast=True, padding_side='right',
truncation_side='right', special_tokens={'unk_token': '[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]', 'cls_token': '[CLS]',
'mask_token': '[MASK]'}, clean_up_tokenization_spaces=False), added_tokens_decoder={
  0: AddedToken("[PAD]", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
  100: AddedToken("[UNK]", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
  101: AddedToken("[CLS]", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
  102: AddedToken("[SEP]", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True),
  103: AddedToken("[MASK]", rstrip=False, lstrip=False, single_word=False, normalized=False, special=True)
})

```

Tokenizer posługuje się słownikiem o stałym rozmiarze. Podowuje to oczywiście, że nie wszystkie wyrazy występujące w tekście, będą się w nim znajdowały. Co więcej, jeśli użyjemy tokenizera do podziału tekstu w innym języku, niż ten dla którego został on stworzony, to taki tekst będzie dzielony na większą liczbę tokenów.

```

sentence1 = tokenizer.encode(
    "The quick brown fox jumps over the lazy dog.", return_tensors="pt"
)
print(sentence1)
print(sentence1.shape)

sentence2 = tokenizer.encode("Zażółć gęślą jaźń.", return_tensors="pt")
print(sentence2)
print(sentence2.shape)

```

```

tensor([[ 101, 1109, 3613, 3058, 17594, 15457, 1166, 1103, 16688, 3676,
          119, 102]])
torch.Size([1, 12])
tensor([[ 101, 163, 1161, 28259, 7774, 20671, 7128, 176, 28221, 28244,
          1233, 28213, 179, 1161, 28257, 19339, 119, 102]])
torch.Size([1, 18])

```

Korzystając z tokenizera dla języka angielskiego do podziału polskiego zdania, widzimy, że otrzymujemy znacznie większą liczbę tokenów. Żeby zobaczyć, w jaki sposób tokenizer dokonał podziału tekstu, możemy wywołać `convert_ids_to_tokens`:

```

print("|".join(tokenizer.convert_ids_to_tokens(list(sentence1[0]))))
print("|".join(tokenizer.convert_ids_to_tokens(list(sentence2[0]))))

[CLS]|The|quick|brown|fox|jumps|over|the|lazy|dog|.|[SEP]
[CLS]|Z|#a|##z|##ó|##ł|##ć|g|##ę|##ś|##ł|##ą|j|##a|##z|##ń|.|[SEP]

```

Widzimy, że dla języka angielskiego wszystkie wyrazy w zdaniu zostały przekształcone w pojedyncze tokeny. W przypadku zdania w języku polskim, zawierającego szereg znaków diakrytycznych sytuacja jest zupełnie inna - każdy znak został wyodrębniony do osobnego sub-tokenu. To, że mamy do czynienia z sub-tokenami sygnalizowane jest przez dwa krzyżyki poprzedzające dany sub-token. Oznaczają one, że ten sub-token musi być sklejony z poprzedzającym go tokenem, aby uzyskać właściwy łańcuch znaków.

▼ Zadanie 1 (0.5 punkt)

Wykorzystaj tokenizer dla modelu `allegro/herbert-base-cased`, aby dokonać tokenizacji tych samych zdań. Jakie wnioski można wyciągnąć przyglądając się sposobowi tokenizacji za pomocą różnych słowników?

```
!pip install sacremoses
```

```

Collecting sacremoses
  Downloading sacremoses-0.1.1-py3-none-any.whl.metadata (8.3 kB)
Requirement already satisfied: regex in /usr/local/lib/python3.10/dist-packages (from sacremoses) (2024.9.11)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from sacremoses) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from sacremoses) (1.4.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from sacremoses) (4.66.6)
Downloading sacremoses-0.1.1-py3-none-any.whl (897 kB)
 897.5/897.5 kB 40.5 MB/s eta 0:00:00
Installing collected packages: sacremoses
Successfully installed sacremoses-0.1.1


```

```

tokenizerAHBC = AutoTokenizer.from_pretrained("allegro/herbert-base-cased")
modelAHBC = AutoModelForMaskedLM.from_pretrained("allegro/herbert-base-cased")

modelAHBC.to(device)

```

	tokenizer_config.json: 100%	229/229 [00:00<00:00, 14.6kB/s]
	config.json: 100%	472/472 [00:00<00:00, 28.6kB/s]
	vocab.json: 100%	907k/907k [00:01<00:00, 573kB/s]
	merges.txt: 100%	556k/556k [00:00<00:00, 2.18MB/s]
	special_tokens_map.json: 100%	129/129 [00:00<00:00, 1.90kB/s]
	pytorch_model.bin: 100%	654M/654M [00:08<00:00, 90.9MB/s]

```
BertForMaskedLM(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(50000, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
)
(cls): BertOnlyMLMHead(
  (predictions): BertLMPredictionHead(
    (transform): BertPredictionHeadTransform(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (transform_act_fn): GELUActivation()
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    )
    (decoder): Linear(in_features=768, out_features=50000, bias=True)
  )
)
```

```
sentence = "Znasz li ten kraj?"
```

```
sentenceTok2 = tokenizerAHBC.encode(sentence, return_tensors="pt")
sentenceTok1 = tokenizer.encode(sentence, return_tensors="pt")
```

```
print("|".join(tokenizer.convert_ids_to_tokens(list(sentenceTok1[0]))))
print("|".join(tokenizerAHBC.convert_ids_to_tokens(list(sentenceTok2[0]))))
```

→ [CLS]|Z|##nas|##z|l|##i|ten|k|##raj|?|[SEP]
 <s>|Z|nasz</w>|li</w>|ten</w>|kraj</w>|?</w>|</s>

Aby edytować zawartość komórki, kliknij ją dwukrotnie (lub naciśnij klawisz Enter)

W wynikach tokenizacji poza wyrazami/tokenami występującymi w oryginalnym tekście pojawiają się jeszcze dodatkowe znaczniki [CLS] oraz [SEP] (albo inne znaczniki - w zależności od użytego słownika). Mają one specjalne znaczenie i mogą być wykorzystywane do realizacji specyficznych funkcji związanych z analizą tekstu. Np. reprezentacja tokenu [CLS] wykorzystywana jest w zadaniach klasyfikacji zdań. Z kolei token [SEP] wykorzystywany jest do odróżnienia zdań, w zadaniach wymagających na wejściu dwóch zdań (np. określenia, na ile zdania te są podobne do siebie).

✓ Modelowanie języka

Modele pretrenowane w reżimie self-supervised learning (SSL) nie posiadają specjalnych zdolności w zakresie rozwiązywania konkretnych zadań z zakresu przetwarzania języka naturalnego, takich jak odpowiadanie na pytania, czy klasyfikacja tekstu (z wyjątkiem bardzo dużych modeli, takich jak np. GPT-3, których model językowy zdolny jest do predykcji np. sensownych odpowiedzi na pytania). Można je jednak wykorzystać do określania prawdopodobieństwa wyrazów w tekście, a tym samym do sprawdzenia, jaką wiedzę posiada określony model w zakresie znajomości języka, czy też ogólną wiedzę o świecie.

Aby sprawdzić jak model radzi sobie w tych zadaniach, możemy dokonać inferencji na danych wejściowych, w których niektóre wyrazy zostaną zastąpione specjalnymi symbolami maskującymi, wykorzystywanymi w trakcie pre-treningu modelu.

Należy mieć na uwadze, że różne modele mogą korzystać z różnych specjalnych sekwencji w trakcie pretreningu. Np. Bert korzysta z sekwencji [MASK]. Wygląd tokenu maskującego lub jego identyfikator możemy sprawdzić w [pliku konfiguracji tokenizera](#) dystrybuowanym razem z modelem, albo odczytać wprost z instancji tokenizera.

W pierwszej kolejności, spróbujemy uzupełnić brakujący wyraz w angielskim zdaniu.

```
sentence_en_text = "The quick brown [MASK] jumps over the lazy dog."

sentence_en = tokenizer.encode(
    sentence_en_text, return_tensors="pt"
)
#
print("|".join(tokenizer.convert_ids_to_tokens(list(sentence_en[0]))))
target = model(sentence_en.to(device))
print(target.logits[0][4])
```

→ [CLS]|The|quick|brown|[MASK]|jumps|over|the|lazy|dog|.|[SEP]
 tensor([-5.3489, -5.6063, -5.1303, ..., -5.9625, -4.1559, -4.5403],
 device='cuda:0', grad_fn=<SelectBackward0>)

Ponieważ zdanie po tokenizowaniu uzupełniane jest znacznikiem [CLS], to zamaskowane słowo znajduje się na 4 pozycji. Wywołanie `target.logits[0][4]` pokazuje tensor z rozkładem prawdopodobieństwa poszczególnych wyrazów, które zostało określone na podstawie parametrów modelu. Możemy wybrać wyrazy, które posiadają największe prawdopodobieństwo, korzystając z wywołania `torch.topk`:

```
import torch

top = torch.topk(target.logits[0][4], 5)
top

→ torch.return_types.topk(
  values=tensor([12.1982, 11.2289, 10.6009, 10.1278, 10.0120], device='cuda:0',
    grad_fn=<TopkBackward0>),
  indices=tensor([ 3676, 1663, 5855, 4965, 21566], device='cuda:0'))
```

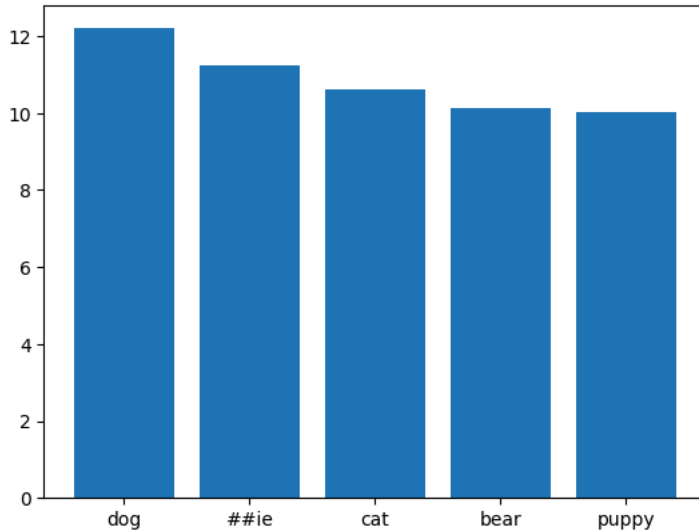
Otrzymaliśmy dwa wektory - `values` zawierający składowe wektora wyjściowego sieci neuronowej (nieznormalizowane) oraz `indices` zawierający indeksy tych składowych. Na tej podstawie możemy wyświetlić wyraz, które według modelu są najbardziej prawdopodobnymi uzupełnieniami zamaskowanego wyrazu:

```
words = tokenizer.convert_ids_to_tokens(top.indices)

import matplotlib.pyplot as plt

plt.bar(words, top.values.cpu().detach().numpy())
```

<BarContainer object of 5 artists>



Według modelu najbardziej prawdopodobnym uzupełnieniem brakującego wyrazu jest `dog` (a nie `fox`). Nieco zaskakujący może być drugi wyraz `##ie`, ale po dodaniu go do istniejącego tekstu otrzymamy zdanie: "The quick brownie jumps over the lazy dog", które również wydaje się sensowne (choć nieco zaskakujące).

✓ Zadanie 2 (1.5 punkty)

Wykorzystując model `allegro/herbert-base-cased` zaproponuj zdania z jednym brakującym wyrazem, weryfikujące zdolność tego modelu do:

- odmiany przez polskie przypadki,
- uwzględniania długodystansowych związków w tekście,
- reprezentowania wiedzy o świecie.

Dla każdego problemu wymyśl po 3 zdania sprawdzające i wyświetl predykcję dla 5 najbardziej prawdopodobnych wyrazów.

Możesz wykorzystać kod z funkcji `plot_words`, który ułatwi Ci wyświetlanie wyników. Zweryfikuj również jaki token maskujący wykorzystywany jest w tym modelu. Pamiętaj również o załadowaniu modelu `allegro/herbert-base-cased`.

Oceń zdolności modelu w zakresie wskazanych zadań.

`tokenizerAHBC.mask_token`

'<mask>'

```
sentenceA = "Zawołałem zmęczonego przechodnia i zapropono <mask> mu jedzenie i pożywienie"
sentence = tokenizerAHBC.encode(sentenceA, return_tensors="pt")
print(" ".join(tokenizerAHBC.convert_ids_to_tokens(list(sentence[0]))))
```

<s>|Za|wo|ł|a|łem</w>|zm|ę|cz|one|go</w>|pr|ech|od|ni|a</w>|i</w>|z|ap|ro|po|no</w>|<mask>|mu</w>|j|ed|ze|nie</w>|i</w>|p|o|ży|wie|nie</w>|</s>

```
def plot_words(sentence, word_model, word_tokenizer, mask="[MASK]"):
    sentence = word_tokenizer.encode(sentence, return_tensors="pt")
    tokens = word_tokenizer.convert_ids_to_tokens(list(sentence[0]))
    print(" ".join(tokens))
    target = word_model(sentence.to(device))
    top = torch.topk(target.logits[0][tokens.index(mask)], 5)
    words = word_tokenizer.convert_ids_to_tokens(top.indices)
    mask_token = word_tokenizer.encode(mask, add_special_tokens=False)[0]
    token_ids = list(sentence[0].cpu().detach().numpy())
    mask_index = token_ids.index(mask_token)
    for word_id in top.indices:
        token_ids[mask_index] = word_id
        print(word_tokenizer.decode(token_ids, skip_special_tokens=True))

plt.xticks(rotation=45)
plt.bar(words, top.values.cpu().detach().numpy())
plt.show()
```

Maska to: <mask>

```
sentenceA = "Zawołałem zmęczonego przechodnia i zapropono<mask> mu jedzenie i pożywienie"
plot_words(sentenceA, modelAHBC, tokenizerAHBC, mask="<mask>")
```

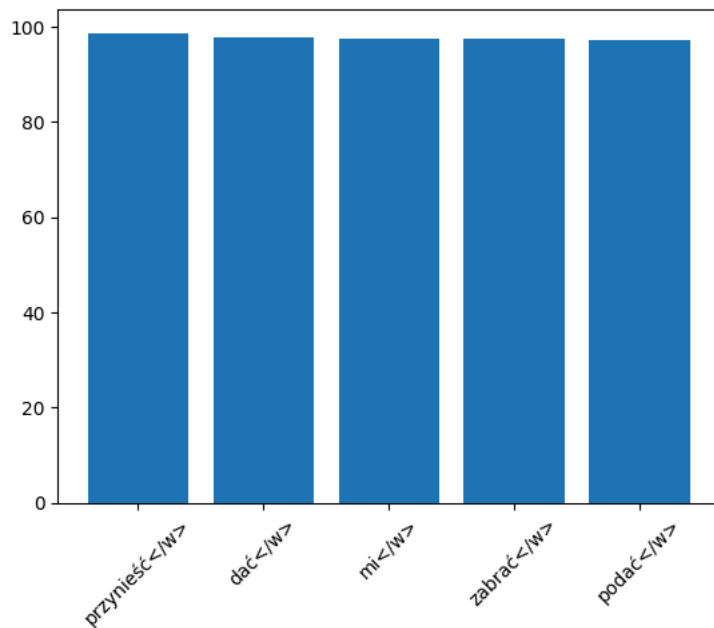
```
sentenceB = "Ja chcę mieć władzę, jaką Ty posiadasz. Ja chcę duszami wład<mask>, jak Ty nimi władasz."  
plot_words(sentenceB, modelAHBC, tokenizerAHBC, mask="<mask>")  
  
sentenceC = "Ziemia krąży wokół <mask>"  
plot_words(sentenceC, modelAHBC, tokenizerAHBC, mask="<mask>")  
  
sentenceD = "Wszechświat ma <mask> lat"  
plot_words(sentenceD, modelAHBC, tokenizerAHBC, mask="<mask>")  
  
sentenceE = "Ludzie żyją około <mask> lat"  
plot_words(sentenceE, modelAHBC, tokenizerAHBC, mask="<mask>")
```



```

<s>|Za|woł|a|łem</w>|zmę|czonego</w>|przechod|nia</w>|i</w>|zapro|po|no</w>|<mask>|mu</w>|jedzenie</w>|i</w>|poży|wienie</w>|</s>
Zawołałem zmęczonego przechodnia i zapropono przynieść mu jedzenie i pożywienie
Zawołałem zmęczonego przechodnia i zapropono dać mu jedzenie i pożywienie
Zawołałem zmęczonego przechodnia i zapropono mi mu jedzenie i pożywienie
Zawołałem zmęczonego przechodnia i zapropono zabrać mu jedzenie i pożywienie
Zawołałem zmęczonego przechodnia i zapropono podać mu jedzenie i pożywienie

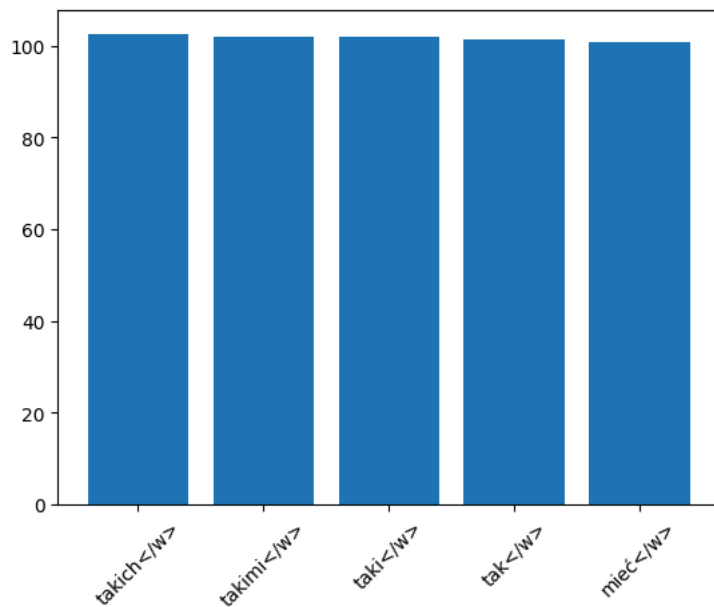
```



```

<s>|Ja</w>|chcę</w>|mieć</w>|władzę</w>|,</w>|jaką</w>|Ty</w>|posiad|asz</w>|. </w>|Ja</w>|chcę</w>|du|szami</w>|wła|d</w>|<mask>|
Ja chcę mieć władzę , jaką Ty posiadasz . Ja chcę duszami wład takich , jak Ty nimi włączasz .
Ja chcę mieć władzę , jaką Ty posiadasz . Ja chcę duszami wład takimi , jak Ty nimi włączasz .
Ja chcę mieć władzę , jaką Ty posiadasz . Ja chcę duszami wład taki , jak Ty nimi włączasz .
Ja chcę mieć władzę , jaką Ty posiadasz . Ja chcę duszami wład tak , jak Ty nimi włączasz .
Ja chcę mieć władzę , jaką Ty posiadasz . Ja chcę duszami wład mieć , jak Ty nimi włączasz .

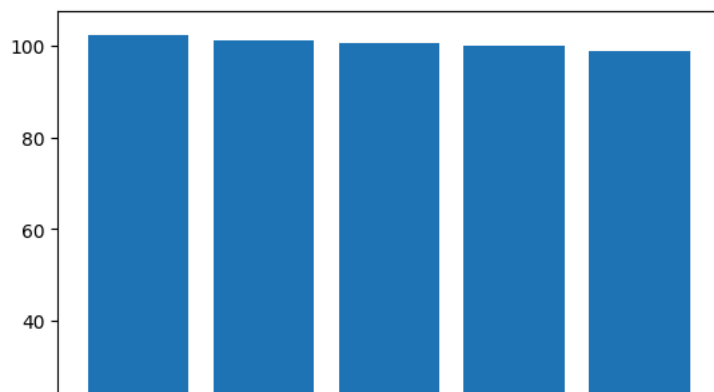
```

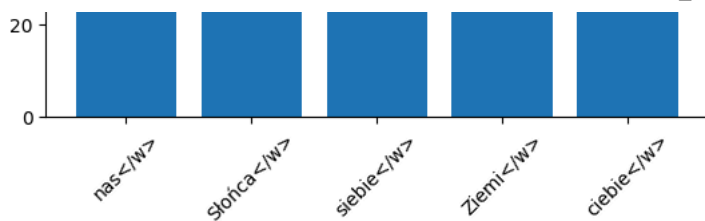


```

<s>|Ziemia</w>|krąży</w>|wokół</w>|<mask>|</s>
Ziemia krąży wokół nas
Ziemia krąży wokół Słońca
Ziemia krąży wokół siebie
Ziemia krąży wokół Ziemi
Ziemia krąży wokół ciebie

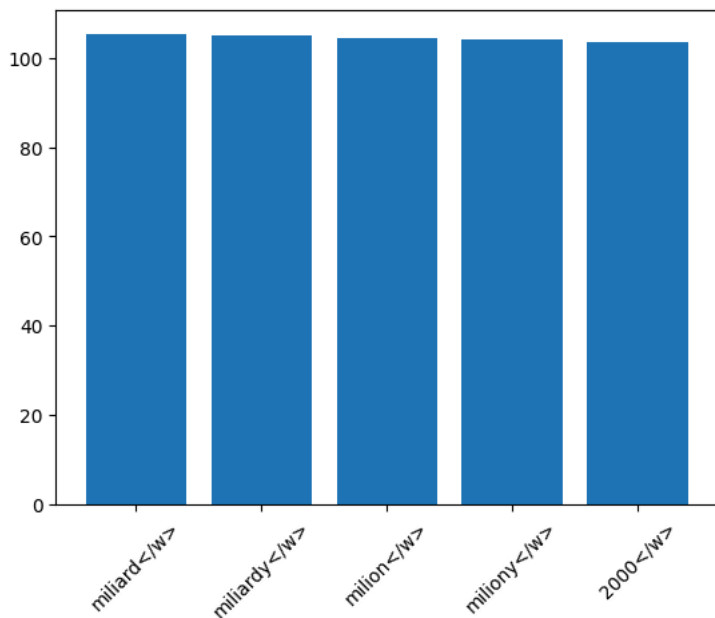
```





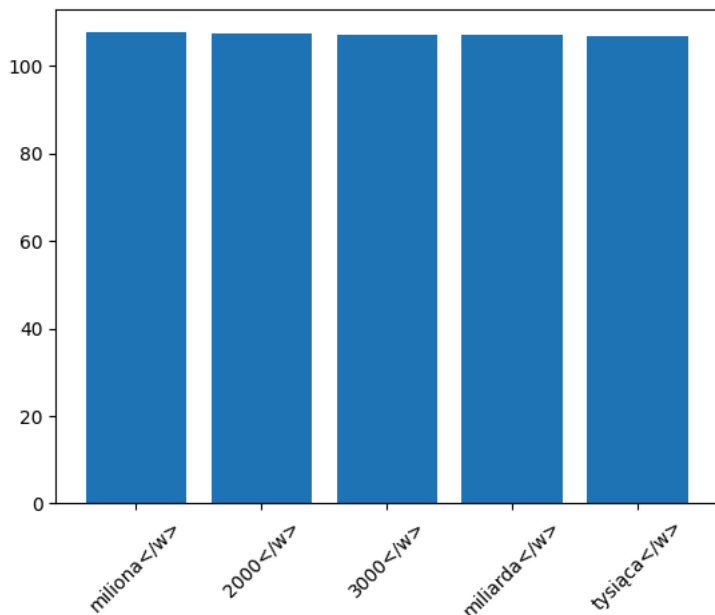
```
<s>|Wszech|świat</w>|ma</w>|<mask>|lat</w>|</s>
```

Wszechświat ma miliard lat
Wszechświat ma miliardy lat
Wszechświat ma milion lat
Wszechświat ma miliony lat
Wszechświat ma 2000 lat



```
<s>|Ludzie</w>|żyją</w>|około</w>|<mask>|lat</w>|</s>
```

Ludzie żyją około miliona lat
Ludzie żyją około 2000 lat
Ludzie żyją około 3000 lat
Ludzie żyją około miliarda lat
Ludzie żyją około tysiąca lat



Aby edytować zawartość komórki, kliknij ją dwukrotnie (lub naciśnij klawisz Enter)

✓ Klasyfikacja tekstu

Pierwszym zadaniem, które zrealizujemy korzystając z modelu HerBERT będzie klasyfikacja tekstu. Będzie to jednak dość nietypowe zadanie. O ile oczekiwanym wynikiem jest klasyfikacja binarna, czyli dość popularny typ klasyfikacji, o tyle dane wejściowe są nietypowe, gdyż są to pary: (pytanie, kontekst). Celem algorytmu jest określenie, czy na zadane pytanie można odpowiedzieć na podstawie informacji znajdujących się w kontekście.

Model tego rodzaju jest nietypowy, ponieważ jest to zadanie z zakresu klasyfikacji par tekstów, ale my potraktujemy je jak zadanie klasyfikacji jednego tekstu, oznaczając jedynie fragmenty tekstu jako `Pytanie:` oraz `Kontekst:`. Wykorzystamy tutaj zdolność modeli transformacyjnych do automatycznego nauczenia się tego rodzaju znaczników, przez co proces przygotowania danych będzie bardzo uproszczony.

Zbiorem danych, który wykorzystamy do treningu i ewaluacji modelu będzie PoQUAD - zbiór inspirowany angielskim [SQuADem](#), czyli zbiorem zawierającym ponad 100 tys. pytań i odpowiadających im odpowiedzi. Zbiór ten powstał niedawno i jest jeszcze rozbudowywany. Zawiera on pytania, odpowiedzi oraz konteksty, na podstawie których można udzielić odpowiedzi.

W dalszej części laboratorium skoncentrujemy się na problemie odpowiadania na pytania.

✓ Przygotowanie danych do klasyfikacji

Przygotowanie danych rozpoczniemy od sklonowania repozytorium zawierającego pytania i odpowiedzi.

```
!pip install datasets
```

```
model.safetensors: 100% 654M/654M [00:26<00:00, 21.0MB/s]

Collecting datasets
  Downloading datasets-3.2.0-py3-none-any.whl.metadata (20 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from datasets) (3.16.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (1.26.4)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (17.0.0)
Collecting dill<0.3.9,>=0.3.0 (from datasets)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.10/dist-packages (from datasets) (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.10/dist-packages (from datasets) (4.66.6)
Collecting xxhash (from datasets)
  Downloading xxhash-3.5.0-cp310-cp310-manylinux_2_17_x86_64_manylinux2014_x86_64.whl.metadata (12 kB)
Collecting multiprocessing<0.70.17 (from datasets)
  Downloading multiprocessing-0.70.16-py310-none-any.whl.metadata (7.2 kB)
Collecting fsspec<=2024.9.0,>=2023.1.0 (from fsspec[http]<=2024.9.0,>=2023.1.0->datasets)
  Downloading fsspec-2024.9.0-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets) (3.11.10)
Requirement already satisfied: huggingface-hub>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.26.5)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from datasets) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (6.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (2.4.4)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: async-timeout<6.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (24.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (6.1.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (0.2.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.18.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.23.0) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets) (3.10)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets) (2024.12.14)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->datasets) (1.17.0)
Downloading datasets-3.2.0-py3-none-any.whl (480 kB)
480.6/480.6 kB 2.6 MB/s eta 0:00:00
Downloading dill-0.3.8-py3-none-any.whl (116 kB)
116.3/116.3 kB 5.6 MB/s eta 0:00:00
Downloading fsspec-2024.9.0-py3-none-any.whl (179 kB)
179.3/179.3 kB 14.4 MB/s eta 0:00:00
Downloading multiprocessing-0.70.16-py310-none-any.whl (134 kB)
134.8/134.8 kB 12.4 MB/s eta 0:00:00
Downloading xxhash-3.5.0-cp310-cp310-manylinux_2_17_x86_64_manylinux2014_x86_64.whl (194 kB)
194.1/194.1 kB 17.3 MB/s eta 0:00:00
Installing collected packages: xxhash, fsspec, dill, multiprocessing, datasets
Attempting uninstall: fsspec
  Found existing installation: fsspec 2024.10.0
  Uninstalling fsspec-2024.10.0:
    Successfully uninstalled fsspec-2024.10.0
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the
gcsfs 2024.10.0 requires fsspec==2024.10.0, but you have fsspec 2024.9.0 which is incompatible.
Successfully installed datasets-3.2.0 dill-0.3.8 fsspec-2024.9.0 multiprocessing-0.70.16 xxhash-3.5.0
```

```
from datasets import load_dataset
```

```
dataset = load_dataset("clarin-pl/poquad")
```

```
README.md: 100% 317/317 [00:00<00:00, 16.8kB/s]
poquad.py: 100% 5.35k/5.35k [00:00<00:00, 154kB/s]
0000.parquet: 100% 12.5M/12.5M [00:00<00:00, 29.2MB/s]
0000.parquet: 100% 1.61M/1.61M [00:00<00:00, 24.1MB/s]
Generating train split: 100% 46187/46187 [00:00<00:00, 72915.73 examples/s]
Generating validation split: 100% 5764/5764 [00:00<00:00, 56528.83 examples/s]
```

Sprawdźmy co znajduje się w zbiorze danych.

```
dataset
```

```
DatasetDict({
  train: Dataset({
    features: ['id', 'title', 'context', 'question', 'answers'],
    num_rows: 46187
  })
  validation: Dataset({
    features: ['id', 'title', 'context', 'question', 'answers'],
    num_rows: 5764
  })
})
```

Zbiór danych jest podzielony na dwie części: treningową i walidacyjną. Rozmiar części treningowej to ponad 46 tysięcy pytań i odpowiedzi, natomiast części walidacyjnej to ponad 5 tysięcy pytań i odpowiedzi.

Dane zbioru przechowywane są w plikach `poquad_train.json` oraz `poquad_dev.json`. Dostarczenie podziału na te grupy danych jest bardzo częstą praktyką w przypadku publicznych, dużych zbiorów danych, gdyż umożliwia porównywanie różnych modeli, korzystając z dokładnie takiego samego zestawu danych. Prawdopodobnie istnieje również zbiór `poquad_test.json`, który jednak nie jest udostępniany publicznie. Tak jest w przypadku SQuADu - twórcy zbioru automatycznie ewaluują dostarczane modele, ale nie udostępniają zbioru testowego. Dzięki temu trudniej jest nadmiernie dopasować model do danych testowych.

Struktura każdej z dostępnych części jest taka sama. Zgodnie z powyższą informacją zawiera ona następujące elementy:

- `id` - identyfikator pary: pytanie - odpowiedź,
- `title` - tytuł artykułu z Wikipedii, na podstawie którego utworzono parę,
- `context` - fragment treści artykułu z Wikipedii, zawierający odpowiedź na pytanie,
- `question` - pytanie,
- `answers` - odpowiedzi.

Możemy wyświetlić kilka początkowych wpisów części treningowej:

```
dataset['train']['question'][:5]
```

```
['Co było powodem powrócenia konceptu porozumieniu monachijskiego?',
 'Pomiędzy jakimi stronami odbyło się zgromadzenie w sierpniu 1942 roku?',
 'O co ubiegali się polscy przedstawiciele podczas spotkania z sierpnia 1942 roku?',
 'Który z dyplomatów sprzeciwił się konceptowi konfederacji w listopadzie '42?',
 'Kiedy oficjalnie doszło do zawarcia porozumienia?']
```

```
dataset['train']['answers'][:5]
```

```
[{'text': ['wymianą listów Ripka - Stroński'], 'answer_start': [117]},
 {'text': ['E. Beneša i J. Masaryka z jednej a Wł. Sikorskiego i E. Raczyńskiego'],
  'answer_start': [197]},
 {'text': ['podpisanie układu konfederacyjnego'], 'answer_start': [315]},
 {'text': ['E. Beneš'], 'answer_start': [558]},
 {'text': ['28 listopada 1942'], 'answer_start': [691]}]
```

Niestety, autorzy zbioru danych, pomimo tego, że dane te znajdują się w źródłowym zbiorze danych, nie udostępniają dwóch ważnych informacji: o tym, czy można odpowiedzieć na dane pytanie oraz jak brzmi generatywna odpowiedź na pytanie. Dlatego póki nie zostanie to naprawione, będziemy dalej pracować z oryginalnymi plikami zbioru danych, które dostępne są na stronie opisującej zbiór danych:

<https://huggingface.co/datasets/clarin-pl/poquad/tree/main>

Pobierz manualnie zbiory `poquad-dev.json` oraz `poquad-train.json`.

```
!wget https://huggingface.co/datasets/clarin-pl/poquad/raw/main/poquad-dev.json
```

```
!wget https://huggingface.co/datasets/clarin-pl/poquad/resolve/main/poquad-train.json
```

```
--2024-12-15 09:33:10-- https://huggingface.co/datasets/clarin-pl/poquad/raw/main/poquad-dev.json
Resolving huggingface.co (huggingface.co)... 13.35.210.114, 13.35.210.77, 13.35.210.61, ...
Connecting to huggingface.co (huggingface.co)|13.35.210.114|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6286317 (6.0M) [text/plain]
Saving to: 'poquad-dev.json'

poquad-dev.json  100%[=====>]  5.99M  1.58MB/s  in 3.8s

2024-12-15 09:33:14 (1.58 MB/s) - 'poquad-dev.json' saved [6286317/6286317]

--2024-12-15 09:33:14-- https://huggingface.co/datasets/clarin-pl/poquad/resolve/main/poquad-train.json
Resolving huggingface.co (huggingface.co)... 13.35.210.114, 13.35.210.77, 13.35.210.61, ...
Connecting to huggingface.co (huggingface.co)|13.35.210.114|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://cdn-lfs.hf.co/repos/18/de/18ded45e8046dd5f58b7365947f5a4298433a0e7710248308670e8cf26059c20/b1ac3acabb49fedb7bb7dbf
--2024-12-15 09:33:15-- https://cdn-lfs.hf.co/repos/18/de/18ded45e8046dd5f58b7365947f5a4298433a0e7710248308670e8cf26059c20/b1ac3acabb49fedb7bb7dbf
Resolving cdn-lfs.hf.co (cdn-lfs.hf.co)... 18.155.68.37, 18.155.68.85, 18.155.68.87, ...
Connecting to cdn-lfs.hf.co (cdn-lfs.hf.co)|18.155.68.37|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 47183344 (45M) [application/json]
Saving to: 'poquad-train.json'

poquad-train.json  100%[=====>]  45.00M  122MB/s  in 0.4s

2024-12-15 09:33:15 (122 MB/s) - 'poquad-train.json' saved [47183344/47183344]
```

Dla bezpieczeństwa, jeśli korzystamy z Google drive, to przeniesiemy pliki do naszego dysku:

```
!mkdir gdrive/MyDrive/poquad
!mv poquad-dev.json gdrive/MyDrive/poquad
!mv poquad-train.json gdrive/MyDrive/poquad
```

```
!head -30 gdrive/MyDrive/poquad/poquad-dev.json
```

```
mkdir: cannot create directory 'gdrive/MyDrive/poquad': File exists
{
  "version": "02-20",
  "data": [
    {
      "id": 9773,
      "title": "Miszna",
      "summary": "Miszna (hebr. מִשְׁנָה miszna „nauczać”, „ustnie przekazywać”, „studiować”, „badać”, od שנה szana „powtarzać”, „różnić”
      "url": "https://pl.wikipedia.org/wiki/Miszna",
      "paragraphs": [
        {
          "context": "Pisma rabiniczne – w tym Miszna – stanowią kompilację poglądów różnych rabinów na określony temat. Zgodnie z v
          "qas": [
            {
              "question": "Czym są pisma rabiniczne?",
              "answers": [
                {
                  "text": "kompilację poglądów różnych rabinów na określony temat",
                  "answer_start": 43,
                  "answer_end": 97,
                  "generative_answer": "kompilacją poglądów różnych rabinów na określony temat"
                }
              ]
            },
            {
              "question": "Z ilu komponentów składała się Tora przekazana Mojżeszowi?",
              "answers": [
                {
                  "text": "dwóch",
                  "answer_start": 172,
```

Struktura pliku odpowiada strukturze danych w zbiorze SQuAD. Dane umieszczone są w kluczu `data` i podzielone na krotki odpowiadające pojedynczym artykułom Wikipedii. W ramach artykułu może być wybranych jeden lub więcej paragrafów, dla których w kluczu `qas` pojawiają się pytania (`question`), flaga `is_impossible`, wskazujące czy można odpowiedzieć na pytanie oraz odpowiedzi (o ile nie jest ustawiona flaga `is_impossible`). Odpowiedzi może być wiele i składają się one z treści odpowiedzi (`text`) traktowanej jako fragment kontekstu, a także naturalnej odpowiedzi na pytanie (`generative_answer`).

Taki podział może wydawać się dziwny, ale zbiór SQuAD zawiera tylko odpowiedzi pierwszego rodzaju. Wynika to z faktu, że w języku angielskim fragment tekstu będzie często stanowił dobrą odpowiedź na pytanie (oczywiście z wyjątkiem pytań dla których odpowiedź to `tak` lub `nie`).

Natomiast ten drugi typ odpowiedzi jest szczególnie przydatny dla języka polskiego, ponieważ często odpowiedź chcemy syntaktycznie dostosować do pytania, co jest niemożliwe, jeśli odpowiedź wskazywana jest jako fragment kontekstu. W sytuacji, w której odpowiedzi były

określane w sposób automatyczny, są one oznaczone jako `plausible_answers`.

✓ Ładowanie danych

```
import json

# Adjust for your needs
# path = "."
path = 'gdrive/MyDrive/poquad'

with open(path + "/poquad-train.json") as input:
    train_data = json.loads(input.read())["data"]

print(f"Train data articles: {len(train_data)}")

with open(path + "/poquad-dev.json") as input:
    dev_data = json.loads(input.read())["data"]

print(f"Dev data articles: {len(dev_data)}")

print(f"Train questions: {sum([len(e['paragraphs'])[0]['qas'] for e in train_data])}")
print(f"Dev questions: {sum([len(e['paragraphs'])[0]['qas'] for e in dev_data])}")
```

↗ Train data articles: 8553
Dev data articles: 1402
Train questions: 41577
Dev questions: 6809

Zacniemy od wczytania danych i wyświetlenia podstawowych statystyk dotyczących ilości artykułów oraz przypisanych do nich pytań.

Ponieważ w pierwszym problemie chcemy stwierdzić, czy na pytanie można udzielić odpowiedzi na podstawie kontekstu, połączymy wszystkie konteksty w jedną tablicę, aby móc losować z niej dane negatywne, gdyż liczba pytań nie posiadających odpowiedzi jest stosunkowo mała, co prowadziłoby utworzenia niezbalansowanego zbioru.

```
all_contexts = [e["paragraphs"][0]["context"] for e in train_data] + [
    e["paragraphs"][0]["context"] for e in dev_data
]
```

W kolejnym kroku zamieniamy dane w formacie JSON na reprezentację zgodną z przyjętym założeniem. Chcemy by kontekst oraz pytanie występowały obok siebie i każdy z elementów był sygnalizowany wyrażeniem: Pytanie: i Kontekst: . Treść klasyfikowanego tekstu przyporządkowujemy do klucza `text`, natomiast klasę do klucza `label`, gdyż takie są oczekiwania biblioteki Transformer.

Pytania, które mają ustawioną flagę `is_impossible` na `True` trafiają wprost do przekształconego zbioru. Dla pytań, które posiadają odpowiedź, dodatkowo losowany jest jeden kontekst, który stanowi negatywny przykład. Weryfikujemy tylko, czy kontekst ten nie pokrywa się z kontekstem, który przypisany był do pytania. Nie przeprowadzamy bardziej zaawansowanych analiz, które pomogłyby wykluczyć sytuację, w której inny kontekst również zawiera odpowiedź na pytanie, gdyż prawdopodobieństwo wylosowania takiego kontekstu jest bardzo małe.

Na końcu wyświetlamy statystyki utworzonego zbioru danych.

```
import random

tuples = [], []


for idx, dataset in enumerate([train_data, dev_data]):
    for data in dataset:
        context = data["paragraphs"][0]["context"]
        for question_answers in data["paragraphs"][0]["qas"]:
            question = question_answers["question"]
            if question_answers["is_impossible"]:
                tuples[idx].append(
                    {
                        "text": f"Pytanie: {question} Kontekst: {context}",
                        "label": 0,
                    }
                )
            else:
                tuples[idx].append(
                    {
                        "text": f"Pytanie: {question} Kontekst: {context}",
                        "label": 1,
                    }
                )
        while True:
```

```

negative_context = random.choice(all_contexts)
if negative_context != context:
    tuples[idx].append(
        {
            "text": f"Pytanie: {question} Kontekst: {negative_context}",
            "label": 0,
        }
    )
    break

train_tuples, dev_tuples = tuples
print(f"Total count in train/dev: {len(train_tuples)}/{len(dev_tuples)}")
print(
    f"Positive count in train/dev: {sum([e['label'] for e in train_tuples])}/{sum([e['label'] for e in dev_tuples])}"
)

```

 Total count in train/dev: 75605/12372
 Positive count in train/dev: 34028/5563


Widzimy, że uzyskane zbiory danych cechują się dość dobrym zbalansowaniem.

Dobrą praktyką po wprowadzeniu zmian w zbiorze danych, jest wyświetlenie kilku przykładowych punktów danych, w celu wykrycia ewentualnych błędów, które powstały na etapie konwersji zbioru. Pozwala to uniknąć nieprzyjemnych niespodzianek, np. stworzenie identycznego zbioru danych testowych i treningowych.

```

print(train_tuples[0:1])
print(dev_tuples[0:1])

```

 [{"text": "Pytanie: Co było powodem powrócenia konceptu porozumienia monachijskiego? Kontekst: Projekty konfederacji zaczęły się zał", "label": 1}, {"text": "Pytanie: Czym są pisma rabiniczne? Kontekst: Pisma rabiniczne - w tym Miszna - stanowią kompilację poglądów różnych rabin", "label": 1}]

Ponieważ mamy nowe zbiory danych, możemy opakować je w klasy ułatwiające manipulowanie nimi. Ma to szczególne znaczenie w kontekście szybkiej tokenizacji tych danych, czy późniejszego szybkiego wczytywania wcześniej utworzonych zbiorów danych.

W tym celu wykorzystamy bibliotekę `datasets`. Jej kluczowymi klasami są `Dataset` reprezentujący jeden z podzbiorów zbioru danych (np. podzbiór testowy) oraz `DatasetDict`, który łączy wszystkie podzbiory w jeden obiekt, którym możemy manipulować w całości. (Gdyby autorzy udostępnili odpowiedni skrypt ze zbiorem, moglibyśmy wykorzystać tę bibliotekę bez dodatkowej pracy).


Dodatkowo zapiszemy tak utworzony zbiór danych na dysku. Jeśli później chcielibyśmy wykorzystać stworzony zbiór danych, to możemy to zrobić za pomocą komendy `load_dataset`.

```

from datasets import Dataset, DatasetDict

train_dataset = Dataset.from_list(train_tuples)
dev_dataset = Dataset.from_list(dev_tuples)
datasets = DatasetDict({"train": train_dataset, "dev": dev_dataset})
datasets.save_to_disk(path + "/question-context-classification")

```

 Saving the dataset (1/1 shards): 100% 75605/75605 [00:05<00:00, 27362.30 examples/s]
 Saving the dataset (1/1 shards): 100% 12372/12372 [00:01<00:00, 5499.82 examples/s]

Dane tekstowe przed przekazaniem do modelu wymagają tokenizacji (co widzieliśmy już wcześniej). Efektywne wykonanie tokenizacji na całym zbiorze danych ułatwione jest przez obiekt `DatasetDict`. Definiujemy funkcję `tokenize_function`, która korzystając z załadowanego tokenizera, zamienia tekst na identyfikatory.

W wywołaniu używamy opcji `padding` - uzupełniamy wszystkie teksty do długości najdłuższego tekstu. Dodatkowo, jeśli któryś tekst wykracza poza maksymalną długość obsługiwaną przez model, to jest on przycinany (`truncation=True`).

Tokenizację aplikujemy do zbioru z wykorzystaniem przetwarzania batchowego (`batched=True`), które pozwala na szybsze stokenizowanie dużego zbioru danych.

```

from transformers import AutoTokenizer

pl_tokenizer = AutoTokenizer.from_pretrained("allegro/herbert-base-cased")

def tokenize_function(examples):
    return pl_tokenizer(examples["text"], padding='do_not_pad', truncation=True)

tokenized_datasets = datasets.map(tokenize_function, batched=True)
tokenized_datasets["train"]

```



```
Map: 100% 75605/75605 [01:10<00:00, 1414.20 examples/s]
Map: 100% 12372/12372 [00:11<00:00, 1104.72 examples/s]
Dataset({
  features: ['text', 'label', 'input_ids', 'token_type_ids', 'attention_mask'],
  num_rows: 75605
})
```

Stokenizowane dane zawierają dodatkowe pola: `input_ids`, `token_type_ids` oraz `attention_mask`. Dla nas najważniejsze jest pole `input_ids`, które zawiera identyfikatory tokenów. Pozostałe dwa pola są ustawione na identyczne wartości (wszystkie tokeny mają ten sam typ, maska atencji zawiera same jedynki), więc nie są one dla nas zbyt interesujące. Zobaczmy pola `text`, `input_ids` oraz `attention_mask` dla pierwszego przykładu:

```
example = tokenized_datasets["train"][0]
print(example["text"])
print("-" * 60)
print(example["input_ids"])
print("-" * 60)
print(example["attention_mask"])
```

[illegible]

Możem też sprawdzić, jak został stokenizowany pierwszy przykład:

```
print("|".join(pl_tokenizer.convert_ids_to_tokens(list(example["input_ids"]))))
```

➡ <s>|Pytanie</w>|:</w>|Co</w>|był</w>|powodem</w>|powró</w>|cenia</w>|koncep</w>|tu</w>|porozumieniu</w>|mona</w>|chi</w>|jskiego</w>|?</w>|Kon</w>|tek</w>|st</w>|

Widzimy, że wyrazy podzielone są sensownie.

Możemy sprawdzić, że liczba tokenów w polu `inut_ids`, które są różne od tokenu wypełnienia (`[PAD] = 1`) oraz maska uwagi, mają tę samą długość:

```
print(len([e for e in example["input_ids"] if e != 1]))
print(len([e for e in example["attention_mask"] if e == 1]))
```

→ 169
169

Mając pewność, że przygotowane przez nas dane są prawidłowe, możemy przystąpić do procesu uczenia modelu.

- ✓ Trening z użyciem transformersów

Biblioteka Transformers pozwala na załadowanie tego samego modelu dostosowanego do różnych zadań. Wcześniej używaliśmy modelu HerBERT do predykcji brakującego wyrazu. Teraz ładujemy ten sam model, ale z inną "głową". Zostanie użyta warstwa, która pozwala na klasyfikację całego tekstu do jednej z n-klas. Wystarczy podmienić klasę, za pomocą której ładujemy model na

AutoModelForSequenceClassification:

```
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained(
    "allegro/herbert-base-cased", num_labels=2
)

model
```

```

[1] Some weights of BertForSequenceClassification were not initialized from the model checkpoint at allegro/herbert-base-cased and are r
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(50000, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
)

```

```

(encoder): BertEncoder(
  (layer): ModuleList(
    (0-11): 12 x BertLayer(
      (attention): BertAttention(
        (self): BertSdpaSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
        (intermediate_act_fn): GELUActivation()
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (pooler): BertPooler(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (activation): Tanh()
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)

```

Aby przyspieszyć trening, będziemy chcieli wybrać tylko niektóre spośród wszystkich trenowalnych parametrów. Wyświetlmy zatem listę nazw dostępnych parametrów, w tym modelu.

```

for name, param in model.named_parameters():
    print(name)

```



```
bert.encoder.layer.11.attention.self.value.bias
bert.encoder.layer.11.attention.output.dense.weight
bert.encoder.layer.11.attention.output.dense.bias
bert.encoder.layer.11.attention.output.LayerNorm.weight
bert.encoder.layer.11.attention.output.LayerNorm.bias
bert.encoder.layer.11.intermediate.dense.weight
bert.encoder.layer.11.intermediate.dense.bias
bert.encoder.layer.11.output.dense.weight
bert.encoder.layer.11.output.dense.bias
bert.encoder.layer.11.output.LayerNorm.weight
bert.encoder.layer.11.output.LayerNorm.bias
bert.pooler.dense.weight
bert.pooler.dense.bias
classifier.weight
classifier.bias
```

▼ Zadanie 3 (0.5 punktu)

Korzystając z atrybutu `requires_grad`, spraw aby następujące parametry:

- klasyfikatory,
- warstwy poolingów,
- ostatniej warstwy encodera,

były jedynymi parametrami podlegającymi uczeniu. Zwróć uwagę na fakt, że domyślnie wszystkie parametry podlegają uczeniu.

```
for param in model.bert.parameters():
    param.requires_grad = False
#

for param in model.classifier.parameters():
    param.requires_grad = True
for param in model.bert.pooler.parameters():
    param.requires_grad = True
for param in model.bert.encoder.layer[-1].parameters():
    param.requires_grad = True

def print_trainable_parameters(model):
    trainable_params = 0
    all_params = 0
    for name, param in model.named_parameters():
        all_params += param.numel()
        if param.requires_grad:
            trainable_params += param.numel()
    trainable = 100 * trainable_params / all_params
    print(
        f"trainable params: {trainable_params:,d} || all params: {all_params:,d} "
        f"|| trainable%: {trainable:.4f}%"
    )
    return trainable

trainable_proportion = print_trainable_parameters(model)

↔ trainable params: 7,680,002 || all params: 124,444,418 || trainable%: 6.1714%

assert 5 < trainable_proportion < 7
print("Solution correct!")

↔ Solution correct!
```

Komunikat diagnostyczny, który pojawia się przy ładowaniu modelu jest zgodny z naszymi oczekiwaniami. Model HerBERT był trenowany do predykcji tokenów, a nie klasyfikacji tekstu. Dlatego też ostatnia warstwa (`classifier.weight` oraz `classifier.bias`) jest inicjowana losowo. Wagi zostaną ustalone w trakcie procesu fine-tuningu modelu.

Jeśli porównamy wersje modeli załadowane za pomocą różnych klas, to zauważymy, że różnią się one tylko na samym końcu. Jest to zgodne z założeniami procesu pre-treningu i fine-tuningu. W pierwszy etapie model uczy się zależności w języku, korzystając z zadania maskowanego modelowania języka (Masked Language Modeling). W drugim etapie model dostosowywane jest do konkretnego zadania, np. klasyfikacji binarnej tekstu.

Korzystanie z biblioteki Transformers uwalnia nas od manualnego definiowania pętli uczącej, czy wywoływania algorytmu wstecznej propagacji błędu. Trening realizowany jest z wykorzystaniem klasy `Trainer` (i jej specjalizacji). Argumenty treningu określone są natomiast w klasie `TrainingArguments`. Klasy te są [bardzo dobrze udokumentowane](#), więc nie będziemy omawiać wszystkich możliwych opcji.

Najważniejsze opcje są następujące:

- `output_dir` - katalog do którego zapisujemy wyniki,
- `do_train` - wymagamy aby przeprowadzony był trening,

- `do_eval` - wymagamy aby przeprowadzona była ewaluacja modelu,
- `evaluation_strategy` - określenie momentu, w którym realizowana jest ewaluacja,
- `evaluation_steps` - określenie co ile kroków (krok = przetworzenie 1 batcha) ma być realizowana ewaluacja,
- `per_device_train/evaluation_batch_size` - rozmiar batcha w trakcie treningu/ewaluacji,
- `learning_rate` - szybkość uczenia,
- `num_train_epochs` - liczba epok uczenia,
- `logging...` - parametry logowania postępów uczenia,
- `save_strategy` - jak często należy zapisywać wytrenowany model,
- `fp16/bf16` - użycie arytmetyki o zmniejszonej dokładności, przyspieszającej proces uczenia. **UWAGA:** użycie niekompatybilnej arytmetyki skutkuje niemożnością nauczenia modelu, co jednak nie daje żadnych innych błędów lub komunikatów ostrzegawczych.

```
from transformers import TrainingArguments
import numpy as np
```

```
arguments = TrainingArguments(
    output_dir=path + "/output",
    do_train=True,
    do_eval=True,
    eval_strategy="steps",
    eval_steps=100,
    per_device_train_batch_size=128,
    per_device_eval_batch_size=256,
    learning_rate=2e-04,
    num_train_epochs=1,
    logging_first_step=True,
    logging_strategy="steps",
    logging_steps=50,
    save_strategy="steps",
    save_steps=100,
    save_total_limit=1,
    metric_for_best_model="accuracy",
    fp16=True,
    lr_scheduler_type="cosine",
    warmup_ratio=0.1,
    seed=42,
    load_best_model_at_end=True,
    label_smoothing_factor=0.1,
    group_by_length=True,
    eval_on_start=True,
)
```

W trakcie treningu będziemy chcieli zobaczyć, czy model poprawnie radzi sobie z postawionym mu problemem. Najlepszym sposobem na podglądanie tego procesu jest obserwowanie wykresów. Model może raportować szereg metryk, ale najważniejsze dla nas będą następujące wartości:

- wartość funkcji straty na danych treningowych - jeśli nie spada w trakcie uczenia, znaczy to, że nasz model nie jest poprawnie skonstruowany lub dane uczące są niepoprawne,
- wartość jednej lub wielu metryk uzyskiwanych na zbiorze walidacyjnym - możemy śledzić wartość funkcji straty na zbiorze ewaluacyjnym, ale warto również wyświetlać metryki, które da się łatwiej zinterpretować; dla klasyfikacji zbalansowanego zbioru danych może to być dokładność (`accuracy`).

Biblioteka Transformers pozwala w zasadzie na wykorzystanie dowolnej metryki, ale szczególnie dobrze współpracuje z metrykami zdefiniowanymi w bibliotece `evaluate` (również autorstwa Huggingface).

Wykorzystanie metryki wymaga od nas zdefiniowania metody, która akceptuje batch danych, który zawierają predykcje (wektory zwrócone na wyjściu modelu) oraz referencyjne wartości - wartości przechowywane w kluczu `label`. Przed obliczeniem metryki konieczne jest "odcyfrowanie" zwróconych wartości. W przypadku klasyfikacji oznacza to po prostu wybranie najbardziej prawdopodobnej klasy i porównanie jej z klasą referencyjną.

Użycie konkretnej metryki realizowane jest za pomocą wywołania `metric.compute`, która akceptuje predykcje (`predictions`) oraz wartości referencyjne (`references`).

```
!pip install evaluate
```

```
Collecting evaluate
  Downloading evaluate-0.4.3-py3-none-any.whl.metadata (9.2 kB)
Requirement already satisfied: datasets>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from evaluate) (3.2.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from evaluate) (1.26.4)
Requirement already satisfied: dill in /usr/local/lib/python3.10/dist-packages (from evaluate) (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from evaluate) (2.2.2)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/dist-packages (from evaluate) (2.32.3)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-packages (from evaluate) (4.66.6)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages (from evaluate) (3.5.0)
Requirement already satisfied: multiprocessing in /usr/local/lib/python3.10/dist-packages (from evaluate) (0.70.16)
Requirement already satisfied: fsspec>=2021.05.0 in /usr/local/lib/python3.10/dist-packages (from fsspec[http]>=2021.05.0->evaluate)
```

```
Requirement already satisfied: huggingface-hub>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from evaluate) (0.26.5)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from evaluate) (24.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from datasets>=2.0.0->evaluate) (3.16.1)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.10/dist-packages (from datasets>=2.0.0->evaluate) (17.0.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets>=2.0.0->evaluate) (3.11.10)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from datasets>=2.0.0->evaluate) (6.0.2)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.7.0->evaluate) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->evaluate) (3.10)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->evaluate) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->evaluate) (2.2)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->evaluate) (2024.12.14)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->evaluate) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->evaluate) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas->evaluate) (2024.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets>=2.0.0->evaluate) (2.4.4)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets>=2.0.0->evaluate) (1.3.1)
Requirement already satisfied: async-timeout<6.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets>=2.0.0->evaluate) (4.0.3)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets>=2.0.0->evaluate) (25.1.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets>=2.0.0->evaluate) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets>=2.0.0->evaluate) (6.0.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets>=2.0.0->evaluate) (0.2.0)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets>=2.0.0->evaluate) (1.18.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->evaluate) (1.17.0)
Downloading evaluate-0.4.3-py3-none-any.whl (84 kB)
```

```
Installing collected packages: evaluate
Successfully installed evaluate-0.4.3
```

```
import evaluate
```

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

```
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)
```



Downloading builder script: 100%

4.20k/4.20k [00:00<00:00, 159kB/s]

Ostatnim krokiem w procesie treningu jest stworzenie obiektu klasy `Trainer`. Akceptuje ona m.in. model, który wykorzystywany jest w treningu, przygotowane argumenty treningu, zbiory do treningu, ewaluacji, czy testowania oraz wcześniej określoną metodę do obliczania metryki na danych ewaluacyjnych.

W przetwarzaniu języka naturalnego dominującym podejściem jest obecnie rozdzielenie procesu treningu na dwa etapy: pre-treining oraz fine-tuning. W pierwszym etapie model trenowany jest w reżimie self-supervised learning (SSL). Wybierane jest zadanie związane najczęściej z modelowaniem języka - może to być kauzalne lub maskowane modelowanie języka.

W *kauzalnym modelowaniu języka* model językowy, na podstawie poprzedzających wyrazów określa prawdopodobieństwo wystąpienia kolejnego wyrazu. W *maskowanym modelowaniu języka* model językowy odgaduje w tekście część wyrazów, która została z niego usunięta.

W obu przypadkach dane, na których trenowany jest model nie wymagają ręcznego oznakowania (tagowania). Wystarczy jedynie posiadać duży korpus danych językowych, aby wytrenować model, który dobrze radzi sobie z jednym z tych zadań. Model tego rodzaju był pokazany na początku laboratorium.

W drugim etapie - fine-tuningu (dostrajaniu modelu) - następuje modyfikacja parametrów modelu, w celu rozwiązania konkretnego zadania. W naszym przypadku pierwszym zadaniem tego rodzaju jest klasyfikacja. Dostroimy zatem model `herbert-base-cased` do zadania klasyfikacji par: pytanie - kontekst.

Wykorzystamy wcześniej utworzone zbiory danych i dodatkowo zmienimy kolejność danych, tak aby uniknąć potencjalnego problemu z korelacją danych w ramach batcha. Wykorzystujemy do tego wywołanie `shuffle`. Za pomocą funkcji `select` możemy wybrać podzbiór przykładów. Jeśli trening trwa u nas wyjątkowo długo, możemy zmienić domyślne wartości na mniejsze.

Ostatnim elementem jest tzw. `data collator`. Dzięki niemu wszystkie przykłady w jednym batchu mają taką samą długość i mogą być przekształcone do tensora.

```
from transformers import Trainer, DataCollatorWithPadding
```

```
seed = 42
train_examples_count = len(tokenized_datasets["train"])
print(train_examples_count)
dev_examples_count = len(tokenized_datasets["dev"])
print(dev_examples_count)

trainer = Trainer(
    model=model,
    args=arguments,
    train_dataset=tokenized_datasets["train"].select(range(train_examples_count)).shuffle(seed=seed),
```

```
eval_dataset=tokenized_datasets["dev"].select(range(dev_examples_count)).shuffle(seed=seed),
compute_metrics=compute_metrics,
data_collator=DataCollatorWithPadding(tokenizer=pl_tokenizer)
)


↻ 75605
12372
```

Zanim uruchomimy trening, załadujemy jeszcze moduł TensorBoard. Nie jest to krok niezbędny. TensorBoard to biblioteka, która pozwala na wyświetlanie w trakcie procesu trening wartości, które wskazują nam, czy model trenuje się poprawnie. W naszym przypadku będzie to `loss` na danych treningowych, `loss` na danych ewaluacyjnych oraz wartość metryki `accuracy`, którą zdefiniowaliśmy wcześniej. Wywołanie tej komórki na początku nie da żadnego efektu, ale można ją odświeżać, za pomocą ikony w menu TensorBoard (ewentualnie włączyć automatyczne odświeżanie). Wtedy w miarę upływu treningu będziemy mieli podgląd, na przebieg procesu oraz osiągnięte wartości interesujących nas parametrów.

Warto zauważyć, że istnieje szereg innych narzędzi do monitorowania eksperymentów z trenowaniem sieci. Wśród nich dużą popularnością cieszą się [WandB](#) oraz [Neptune.AI](#). Ich zaletą jest m.in. to, że możemy łatwo archiwizować przeprowadzone eksperymenty, porównywać je ze sobą, analizować wpływ hiperparametrów na uzyskane wyniki, itp.

Jeśli wyniki są niewidoczne, otwórz ręcznie adres, np. <http://localhost:6006> jeśli uruchamiasz notebooka lokalnie.

```
!mkdir -p ./output/runs
%load_ext tensorboard
##%tensorboard --logdir gdrive/MyDrive/poquad/output/runs
%tensorboard --logdir ./output/runs
```



TensorBoard

INACTIVE

No dashboards are active for the current data set.

Probable causes:

- You haven't written any data to your event files.
- TensorBoard can't find your event files.

If you're new to using TensorBoard, and want to find out how to add data and set up your event files, check out the [README](#) and perhaps the [TensorBoard tutorial](#).


If you think TensorBoard is configured properly, please see [the section of the README devoted to missing data problems](#) and consider filing an issue on GitHub.

Last reload: Dec 15, 2024, 10:35:29 AM

Log directory: ./output/runs

Uruchomienie procesu treningu jest już bardzo proste, po tym jak przygotowaliśmy wszystkie niezbędne szczegóły. Wystarczy wywołać metodę `trainer.train()`. Warto mieć na uwadze, że proces ten będzie jednak długotrwały - jedna epoka treningu na przygotowanych danych będzie trwała ponad 1 godzinę. Na szczęście, dzięki ustawieniu ewaluacji co 300 kroków, będziemy mogli obserwować jak model radzi sobie z postawionym przed nim problemem na danych ewaluacyjnych.

```
# 3m @ 4080
# trainer.train()
```


 **wandb:** **WARNING** The `run_name` is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please **wandb:** Using wandb-core as the SDK backend. Please refer to <https://wandb.me/wandb-core> for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: <https://wandb.me/wandb-server>)
wandb: You can find your API key in your browser here: <https://wandb.ai/authorize>
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit:
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
Tracking run with wandb version 0.18.7
Run data is saved locally in /content/wandb/run-20241215_093905-bog4ce9y
Syncing run [gdrive/MyDrive/poquad/output](#) to [Weights & Biases \(docs\)](#)
View project at <https://wandb.ai/pawlowiczf-agh/huggingface>
View run at <https://wandb.ai/pawlowiczf-agh/huggingface/runs/bog4ce9y>
Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
[591/591 09:59, Epoch 1/1]

Step	Training Loss	Validation Loss	Accuracy
0	No log	0.733189	0.550356
100	0.513600	0.431930	0.866796
200	0.407600	0.409321	0.868493
300	0.386900	0.385548	0.884821
400	0.387900	0.379870	0.887407
500	0.382900	0.376306	0.889589

Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
Trainer.tokenizer is now deprecated. You should use Trainer.processing_class instead.
TrainOutput(global_step=591, training_loss=0.42661479523944373, metrics={'train_runtime': 730.1558, 'train_samples_per_second': 103.546, 'train_steps_per_second': 0.809, 'total_flos': 8557864881101880.0, 'train_loss': 0.42661479523944373, 'epoch': 1.0})

```
trainer.save_model(path + '/model')
```

```
model = AutoModelForSequenceClassification.from_pretrained(path + '/model')
model.to(device)
```

 BertForSequenceClassification(
 (bert): BertModel(
 (embeddings): BertEmbeddings(
 (word_embeddings): Embedding(50000, 768, padding_idx=1)
 (position_embeddings): Embedding(514, 768)
 (token_type_embeddings): Embedding(2, 768)
 (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
 (dropout): Dropout(p=0.1, inplace=False)
)
 (encoder): BertEncoder(
 (layer): ModuleList(
 (0-11): 12 x BertLayer(
 (attention): BertAttention(
 (self): BertSdpaSelfAttention(
 (query): Linear(in_features=768, out_features=768, bias=True)
 (key): Linear(in_features=768, out_features=768, bias=True)
 (value): Linear(in_features=768, out_features=768, bias=True)
 (dropout): Dropout(p=0.1, inplace=False)
)
 (output): BertSelfOutput(
 (dense): Linear(in_features=768, out_features=768, bias=True)
 (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
 (dropout): Dropout(p=0.1, inplace=False)
)
)
 (intermediate): BertIntermediate(
 (dense): Linear(in_features=768, out_features=3072, bias=True)
 (intermediate_act_fn): GELUActivation()
)
 (output): BertOutput(
 (dense): Linear(in_features=3072, out_features=768, bias=True)
 (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
 (dropout): Dropout(p=0.1, inplace=False)
)
)
)
)
 (pooler): BertPooler(
 (dense): Linear(in_features=768, out_features=768, bias=True)
 (activation): Tanh()
)
)
 (dropout): Dropout(p=0.1, inplace=False)

```
(classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

✓ Zadanie 4 (0.5 punkt)

Wybierz losową stronę z Wikipedii i skopiuj fragment tekstu do Notebook. Zadać 3 pytania, na które można udzielić odpowiedzi na podstawie tego fragmentu tekstu oraz 3 pytania, na które nie można udzielić odpowiedzi. Oceń jakość predykcji udzielanych przez model.

Pamiętaj, aby przełączyć model w tryb inferencji (`model.eval()`). W przeciwnym razie wyniki będą losowe, ponieważ aktywny będzie mechanizm *dropout*.

```
wikiText = 'Szczepionka przeciwtyfusowa prof. Rudolfa Weigla w Muzeum Historii Żydów Polskich w Warszawie. Profesor Weigl nigdy nie wyp:
model.eval()
```

```
questions = [
    "Czy prof. Rudolf Weigl otrzymał nagrodę Nobla?",
    "Czy Niemcy poparli kandydaturę do nagrody Nobla prof. Rudofla Weigla?",

    "Czy prof. Rudolf Weigl umiał jeździć na rowerze?",
    "Czy prof. Rudolf Weigl opracował szczepionkę na COVID-19?",
    "Jak miał na imię ojciec profesora?",
    "Czy Andrzej Duda jest prezydentem Polski?",
    "Ile wynosi dziura budżetowa Polski za rządów Tuska?"
]

for question in questions:

    inputs = pl_tokenizer.encode(f"Pytanie: {question} Kontekst: {wikiText}", return_tensors="pt")

    with torch.no_grad():
        logits = model(inputs.to(device)).logits
        predictions = torch.argmax(logits)
        print(f"Pytanie: {question}")
        print(f"Kontekst: {predictions.item()}")
    #
# end 'for' loop
```

```
Pytanie: Czy prof. Rudolf Weigl otrzymał nagrodę Nobla?
Kontekst: 1
Pytanie: Czy Niemcy poparli kandydaturę do nagrody Nobla prof. Rudofla Weigla?
Kontekst: 1
Pytanie: Czy prof. Rudolf Weigl umiał jeździć na rowerze?
Kontekst: 1
Pytanie: Czy prof. Rudolf Weigl opracował szczepionkę na COVID-19?
Kontekst: 1
Pytanie: Jak miał na imię ojciec profesora?
Kontekst: 1
Pytanie: Czy Andrzej Duda jest prezydentem Polski?
Kontekst: 0
Pytanie: Ile wynosi dziura budżetowa Polski za rządów Tuska?
Kontekst: 0
```

Aby edytować zawartość komórki, kliknij ją dwukrotnie (lub naciśnij klawisz Enter)

✓ Odpowiadanie na pytania

Drugim problemem, którym zajmie się w tym laboratorium jest odpowiadanie na pytania. Zmierzymy się z wariantem tego problemu, w którym model sam formułuje odpowiedź, na podstawie pytania i kontekstu, w których znajduje się odpowiedź na pytanie (w przeciwieństwie do wariantu, w którym model wskazuje lokalizację odpowiedzi na pytanie).

✓ Zadanie 5 (1 punkt)

Rozpocznij od przygotowania danych. Wybierzem tylko te pytania, które posiadają odpowiedź (`is_impossible=False`). Uwzględnij zarówno pytania *pewne* (pole `answers`) jak i *prawdopodobne* (pole `plausible_answers`). Wynikowy zbiór danych powinien mieć identyczną strukturę, jak w przypadku zadania z klasyfikacją, ale etykiety zamiast wartości 0 i 1, powinny zawierać odpowiedź na pytanie, a sama nazwa etykiety powinna być zmieniona z `label` na `labels`, w celu odzwierciedlenia faktu, że teraz zwracane jest wiele etykiet.

Wyświetl liczbę danych (par: pytanie - odpowiedź) w zbiorze treningowym i zbiorze ewaluacyjnym.

Opakuj również zbiory w klasy z biblioteki `datasets` i zapisz je na dysku.

```
!pip install datasets
```

```
Requirement already satisfied: datasets in /usr/local/lib/python3.10/dist-packages (3.2.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from datasets) (3.16.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (1.26.4)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (17.0.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.10/dist-packages (from datasets) (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.10/dist-packages (from datasets) (4.66.6)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages (from datasets) (3.5.0)
Requirement already satisfied: multiprocessing<0.70.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2024.9.0,>=2023.1.0 in /usr/local/lib/python3.10/dist-packages (from fsspec[http]<=2024.9.0,
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets) (3.11.10)
Requirement already satisfied: huggingface-hub>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.26.5)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from datasets) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (6.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (2.4.4)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: async-timeout<6.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (24.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (6.1.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (0.2.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.18.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.23.0->
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets) (2.2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets) (2024
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->datasets)
```

```
import random
from datasets import Dataset, DatasetDict

tuples = [], []

for idx, dataset in enumerate([train_data, dev_data]):
    for data in dataset:
        context = data["paragraphs"][0]["context"]
        for question_answers in data["paragraphs"][0]["qas"]:

            question = question_answers["question"]

            if question_answers["is_impossible"]: break

            answers = question_answers["answers"]

            tuples[idx].append(
                {
                    "text": f"Pytanie: {question} Kontekst: {context}",
                    "labels": answers[0]["text"]
                }
            )

# end 'for' loop

train_tuples, dev_tuples = tuples

train_dataset = Dataset.from_list(train_tuples)
dev_dataset = Dataset.from_list(dev_tuples)
datasets = DatasetDict({"train": train_dataset, "dev": dev_dataset})
datasets.save_to_disk(path + "/question-context-response")
```

```
Saving the dataset (1/1 shards): 100% 26670/26670 [00:00<00:00, 165569.65 examples/s]
Saving the dataset (1/1 shards): 100% 4364/4364 [00:00<00:00, 77375.15 examples/s]
```

Zanim przejdziemy do dalszej części, sprawdźmy, czy dane zostały poprawnie utworzone. Zweryfikujmy przede wszystkim, czy klucze `text` oraz `label` zawierają odpowiednie wartości:

```
print(datasets["train"][0])
```

```
{'text': 'Pytanie: Co było powodem powrócenia konceptu porozumieniu monachijskiego? Kontekst: Projekty konfederacji zaczęły się zał...
```

```
print(datasets["train"][0]["text"])
print(datasets["train"][0]["labels"])
print(datasets["dev"][0]["text"])
print(datasets["dev"][0]["labels"])
```

Pytanie: Co było powodem powrócenia konceptu porozumieniu monachijskiego? Kontekst: Projekty konfederacji zaczęły się załamywać 5 si wymianą listów Ripka - Stroński
 Pytanie: Czym są pisma rabiniczne? Kontekst: Pisma rabiniczne - w tym Miszna - stanowią kompilację poglądów różnych rabinów na okres kompilację poglądów różnych rabinów na określony temat

Tokenizacja danych dla problemu odpowiadania na pytania jest nieco bardziej problematyczna. W pierwszej kolejności trzeba wziąć pod uwagę, że dane wynikowe (etykiety), też muszą podlegać tokenizacji. Realizowane jest to poprzez wywołanie tokenizera, z opcją `text_target` ustawioną na łańcuch, który ma być stokenizowany.

Ponadto wcześniej nie przejmowaliśmy się za bardzo tym, czy wykorzystywany model obsługuje teksty o założonej długości. Teraz jednak ma to duże znaczenie. Jeśli użyjemy modelu, który nie jest w stanie wygenerować odpowiedzi o oczekiwanej długości, to nie możemy oczekiwać, że model ten będzie dawał dobre rezultaty dla danych w zbiorze treningowym i testowym.

W pierwszej kolejności dokonamy więc tokenizacji bez ograniczeń co do długości tekstu. Ponadto, stokenizowane odpowiedzi przypiszemy do klucza `label`. Do tokenizacji użyjemy tokenizera stowarzyszonego z modelem `allegro/plt5-base`.

```
from transformers import AutoTokenizer

plt5_tokenizer = AutoTokenizer.from_pretrained("allegro/plt5-base")

def preprocess_function(examples):
    model_inputs = plt5_tokenizer(examples["text"])
    labels = plt5_tokenizer(text_target=examples["labels"])
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
```

```
tokenized_datasets = datasets.map(preprocess_function, batched=True)
```

```
tokenizer_config.json: 100%          141/141 [00:00<00:00, 9.97kB/s]
config.json: 100%                    658/658 [00:00<00:00, 41.0kB/s]
spiece.model: 100%                   1.12M/1.12M [00:00<00:00, 26.9MB/s]
special_tokens_map.json: 100%        65.0/65.0 [00:00<00:00, 4.15kB/s]
You are using the default legacy behaviour of the <class 'transformers.models.t5.tokenization_t5.T5Tokenizer'>. This is expected, ar
Map: 100%                             26670/26670 [00:21<00:00, 1373.66 examples/s]
Map: 100%                             4364/4364 [00:05<00:00, 714.31 examples/s]
```

Sprawdźmy jak dane wyglądają po tokenizacji:

```
print(tokenized_datasets["train"][0].keys())
print(tokenized_datasets["train"][0]["input_ids"])
print(tokenized_datasets["train"][0]["labels"])
print(len(tokenized_datasets["train"][0]["input_ids"]))
print(len(tokenized_datasets["train"][0]["labels"]))
example = tokenized_datasets["train"][0]

print("".join(plt5_tokenizer.convert_ids_to_tokens(list(example["input_ids"]))))
print("".join(plt5_tokenizer.convert_ids_to_tokens(list(example["labels"]))))

dict_keys(['text', 'labels', 'input_ids', 'attention_mask'])
[21584, 291, 639, 402, 11586, 292, 23822, 267, 1269, 8741, 280, 24310, 42404, 305, 373, 1525, 15643, 291, 2958, 273, 19605, 6869, 27
[39908, 20622, 2178, 18204, 308, 8439, 2451, 1]
165
8
Pytanie:|_Co|_było|_powodem|_po|wrócenia|_kon|cept|_u|_porozumieniu|_monachijski|ego|?|_Kon|tekst|:_|_Projekt|_y|_konfederacji|_z:
_wymianą|_listów|_Ri|pka|_|_Stro|ński|</s>
```

Wykorzystywany przez nas model ma złożoność pamięciową kwadratową ze względu na długość tekstu. Z tego względu chcemy ograniczyć długość danych wejściowych oraz tekstów podlegających predykcji.

✓ Zadanie 6 (0.5 punkt)

Stwórz histogramy prezentujące rozkład długości (jako liczby tokenów) tekstów wejściowych (`input_ids`) oraz odpowiedzi (`labels`) dla zbioru treningowego. Zinterpretuj (skomentuj) otrzymane wyniki.

```
print(tokenized_datasets["train"][0])
```

```
↵ 8, 20622, 2178, 18204, 308, 8439, 2451, 1], 'input_ids': [21584, 291, 639, 402, 11586, 292, 23822, 267, 1269, 8741, 280, 24310, 4246
```

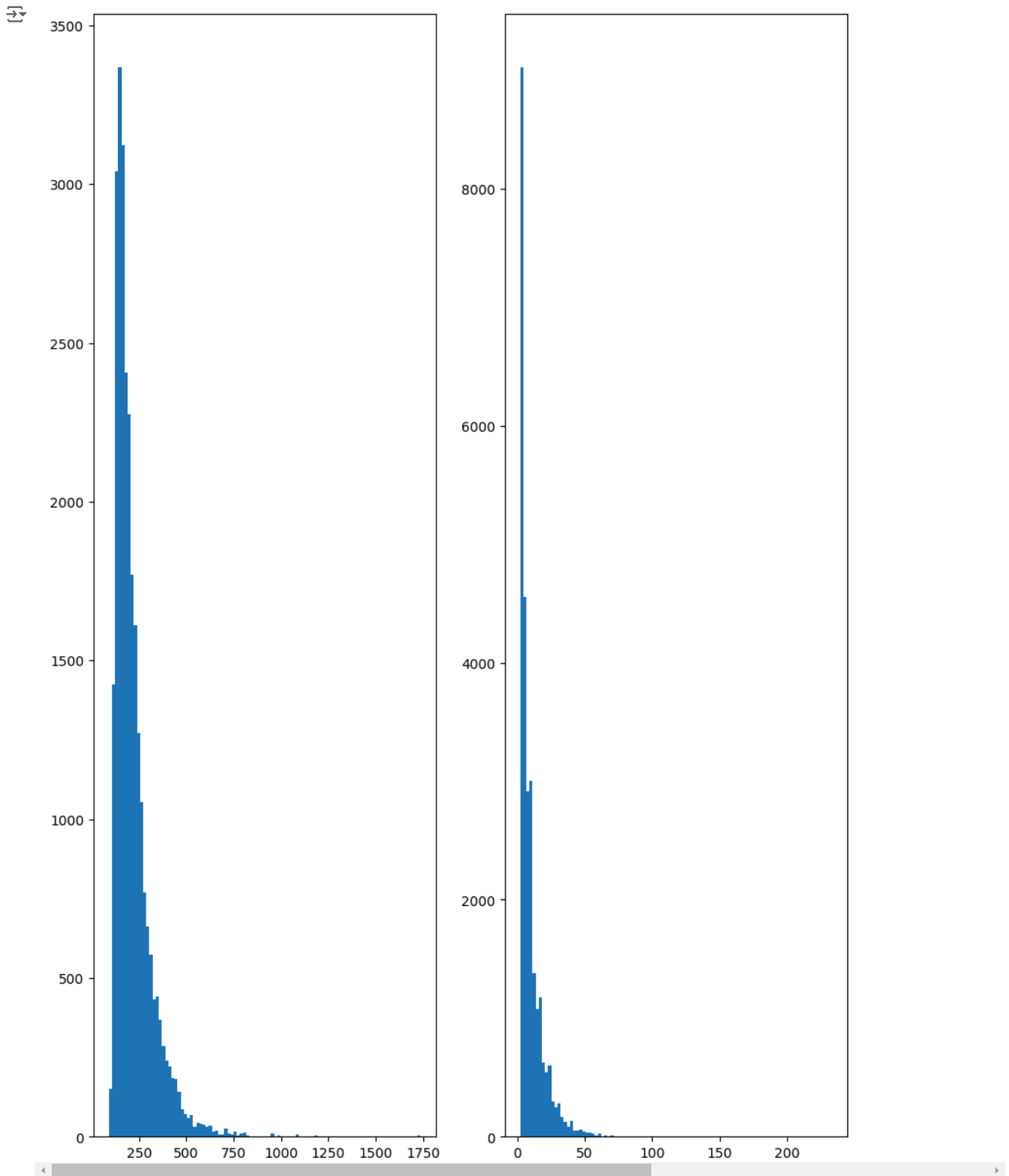
```
import matplotlib.pyplot as plt
import numpy as np
```

```
_, axs = plt.subplots(1, 2, figsize=(10, 15))
```

```
input_ids_list = [len(example["input_ids"]) for example in tokenized_datasets["train"]]
labels_list    = [len(example["labels"]) for example in tokenized_datasets["train"]]
```

```
# for data in tokenized_datasets["train"]:
#     input_ids_list.extend(data["input_ids"])
#     labels_list.extend(data["labels"])
#
```

```
axs[0].hist(input_ids_list, bins=100)
axs[1].hist(labels_list, bins=100)
plt.show()
```



Aby edytować zawartość komórki, kliknij ją dwukrotnie (lub naciśnij klawisz Enter)

Przyjmijmy założenie, że teksty wejściowe będą miały maksymalnie 256 tokenów, a odpowiedzi do długości 32 tokenów.

W poniższym kodzie uwzględniamy również fakt, że przy obliczaniu funkcji straty nie interesuje nas wliczanie tokenów wypełnienia (PAD), gdyż ich udział byłby bardzo duży, a nie wpływają one w żaden pozytywny sposób na ocenę poprawności działania modelu.

Konteksty (pytanie + kontekst odpowiedzi) ograniczamy do 256 tokenów, ze względu na ograniczenia pamięciowe (zajętość pamięci dla modelu jest proporcjonalna do kwadratu długości tekstu). Dla kontekstów nie używamy parametru `padding`, ponieważ w trakcie treningu użyjemy modułu, który automatycznie doda padding, tak żeby wszystkie sekwencje miały długość najdłuższego tekstu w ramach paczki (moduł ten to `DataCollatorForSeq2Seq`).

```
def preprocess_function(examples):
    result = plt5_tokenizer(examples["text"], truncation=True, max_length=256)
```

```

targets = plt5_tokenizer(
    examples["labels"], truncation=True, max_length=32, padding=True
)
target_ids = [
    [(1 if l != plt5_tokenizer.pad_token_id else -100) for l in e]
    for e in targets["input_ids"]
]
result["labels"] = target_ids
return result

```

```
tokenized_datasets = datasets.map(preprocess_function, batched=True)
```

```

Map: 100% 26670/26670 [00:25<00:00, 1186.29 examples/s]
Map: 100% 4364/4364 [00:03<00:00, 1295.65 examples/s]

```

Następnie weryfikujemy, czy przetworzone teksty mają poprawną postać.

```

print(tokenized_datasets["train"][0].keys())
print(tokenized_datasets["train"][0]["input_ids"])
print(tokenized_datasets["train"][0]["labels"])
print(len(tokenized_datasets["train"][0]["input_ids"]))
print(len(tokenized_datasets["train"][0]["labels"]))

```

```

dict_keys(['text', 'labels', 'input_ids', 'attention_mask'])
[21584, 291, 639, 402, 11586, 292, 23822, 267, 1269, 8741, 280, 24310, 42404, 305, 373, 1525, 15643, 291, 2958, 273, 19605, 6869, 2,
39908, 20622, 2178, 18204, 308, 8439, 2451, 1, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100, -100,
165
32

```

Ładowanie modelu

Dla problemu odpowiadania na pytania potrzebować będziemy innego pre-trenowanego modelu oraz innego przygotowania danych. Jako model bazowy wykorzystamy polski wariant modelu T5 - [pT5](#). Model ten trenowany był w zadaniu *span corruption*, czyli zadani polegającym na usunięciu fragmentu tekstu. Model na wejściu otrzymywał tekst z pominiętymi pewnymi fragmentami, a na wyjściu miał odtwarzać te fragmenty. Oryginalny model T5 dodatkowo pretrenowany był na kilku konkretnych zadaniach z zakresu NLP (w tym odpowiadaniu na pytania). W wariacie pT5 nie przeprowadzono jednak takiego dodatkowego procesu.

Poniżej ładujemy model dla zadania, w którym model generuje tekst na podstawie innego tekstu (tzn. jest to zadanie zamiany tekstu na tekst, po angielsku zwanego też *Sequence-to-Sequence*).

```

from transformers import AutoModelForSeq2SeqLM

model = AutoModelForSeq2SeqLM.from_pretrained("allegro/pT5-base")

```

```

pytorch_model.bin: 100% 1.10G/1.10G [00:09<00:00, 172MB/s]

```

W celu poprawy szybkości treningu moglibyśmy użyć podobnej metody jak przy klasyfikacji. Istnieją jednak bardziej efektywne metody, np. low-rank adaptation (LoRA), które dokomponują macierze wag na dwie macierze o mniejszej liczbie parametrych. Są one szczególnie istotne dla uczenia dużych modeli. Dzięki bibliotece PEFT ich użycie jest bardzo proste. Skorzystamy zatem z LoRA przy tworzeniu modelu QA.

W pierwszej kolejności konfigujemy metodę:

```

from peft import LoraConfig, get_peft_model, TaskType

lora_config = LoraConfig(
    r=32, # Rank
    lora_alpha=32,
    target_modules=["q", "v"],
    lora_dropout=0.05,
    bias="none",
    task_type=TaskType.SEQ_2_SEQ_LM # FLAN-T5
)


```

Teraz opakowujemy oryginalny model w model PEFT. Oryginalny model nie będzie modyfikowany. Musimy jednak pamiętać żeby wszędzie używać modelu PEFT.

```

peft_model = get_peft_model(model, lora_config)
print_trainable_parameters(peft_model)

```

 trainable params: 3,538,944 || all params: 278,641,920 || trainable%: 1.2701%
1.2700687678293345

Widzimy, że liczba modyfikowalnych parametrów jest bardzo mała względem oryginalnego modelu.

✓ Trening modelu QA

Ostatnim krokiem przed uruchomieniem treningu jest zdefiniowanie metryk, wskazujących jak model radzi sobie z problemem. Wykorzystamy dwie metryki:

- *exact match* - która sprawdza dokładne dopasowanie odpowiedzi do wartości referencyjnej, metryka ta jest bardzo restrykcyjna, ponieważ pojedynczy znak będzie powodował, że wartość będzie niepoprawna,
- *blue score* - metryka uwzględniająca częściowe dopasowanie pomiędzy odpowiedzią a wartością referencyjną, najczęściej używana jest do oceny maszynowego tłumaczenia tekstu, ale może być również przydatna w ocenie wszelkich zadań, w których generowany jest tekst.

Wykorzystujemy bibliotekę `evaluate`, która zawiera definicje obu metryk.

Przy konwersji identyfikatorów tokenów na tekstu zamieniamy również z powrotem tokeny o wartości -100 na identyfikatory paddingu. W przeciwnym razie dostaniemy błąd o nieistniejącym identyfikatorze tokenu.

W procesie treningu pokazujemy również różnicę między jedną wygenerowaną oraz prawdziwą odpowiedzią dla zbioru ewaluacyjnego. W ten sposób możemy śledzić co rzeczywiście dzieje się w modelu.

```
from transformers import Seq2SeqTrainer, Seq2SeqTrainingArguments
import numpy as np
import evaluate


exact = evaluate.load("exact_match")
bleu = evaluate.load("bleu")

def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.where(predictions != -100, predictions, plt5_tokenizer.pad_token_id)
    decoded_preds = plt5_tokenizer.batch_decode(predictions, skip_special_tokens=True)
    labels = np.where(labels != -100, labels, plt5_tokenizer.pad_token_id)
    decoded_labels = plt5_tokenizer.batch_decode(labels, skip_special_tokens=True)
    print("prediction: " + decoded_preds[0])
    print("reference : " + decoded_labels[0])

    result = exact.compute(predictions=decoded_preds, references=decoded_labels)
    result = {**result, **bleu.compute(predictions=decoded_preds, references=decoded_labels)}
    del result["precisions"]

    prediction_lens = [np.count_nonzero(pred != plt5_tokenizer.pad_token_id) for pred in predictions]
    result["gen_len"] = np.mean(prediction_lens)

    return result
```

 Downloading builder script: 100% 5.67k/5.67k [00:00<00:00, 238kB/s]
Downloading builder script: 100% 5.94k/5.94k [00:00<00:00, 365kB/s]
Downloading extra modules: 4.07k/? [00:00<00:00, 166kB/s]