взять новостные данные из
https://github.com/natasha/corus (https://github.com/natasha/corus)
load_lenta2
нам понадобиться сам текст и заголовок
обучить модель T5/ или GPT для генерации заголовков для статей

In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
# import os
# for dirname, _, filenames in os.walk('/kaggle/input'):
#     for filename in filenames:
#         print(os.path.join(dirname, filename))
```

In [3]:

```
# !wget https://github.com/yutkin/Lenta.Ru-News-Dataset/releases/download/v1.1/lenta-ru-new
```

In [4]:

```
from corus import load_lenta2

path = 'lenta-ru-news.csv.bz2'
records = load_lenta2(path)
next(records)
```

Out[4]:

```
LentaRecord(
    url='https://lenta.ru/news/1914/09/16/hungarnn/',
    title='1914. Русские войска вступили в\xa0пределы Венгрии  ',
    text='Бои у Сопоцкина и Друскеник закончились отступлением германцев. Не
приятель, приблизившись с севера к Осовцу начал артиллерийскую борьбу с креп
остью. В артиллерийском бою принимают участие тяжелые калибры. С раннего утр
а 14 сентября огонь достиг значительного напряжения. Попытка германской пехо
ты пробиться ближе к крепости отражена. В Галиции мы заняли Дембицу. Большая
колонна, отступавшая по шоссе от Перемышля к Саноку, обстреливалась с высот
нашей батареей и бежала, бросив парки, обоз и автомобили. Вылазки гарнизона
Перемышля остаются безуспешными. При продолжающемся отступлении австрийцев о
бнаруживается полное перемешивание их частей, захватываются новые партии пле
нных, орудия и прочая материальная часть. На перевале Ужок мы разбили неприя
тельский отряд, взяли его артиллерию и много пленных и, продолжая преследова
ть, вступили в пределы Венгрии. «Русский инвалид», 16 сентября 1914 года.',
    topic='Библиотека',
    tags='Первая мировая',
    date=datetime.datetime(1914, 9, 16, 0, 0)
)
```

In [5]:

```
def load_lenta_to_list(path, max_number=None):
    records = load_lenta2(path)
    texts, titles = [], []
    for i, record in enumerate(records):
        texts.append(record.text)
        titles.append(record.title)
        if not max_number is None:
            if i >= max_number-1:
                break
    return texts, titles
```

In [6]:

```
texts, titles = load_lenta_to_list(path, max_number=20000)
print(len(texts))
print(len(titles))
```

```
20000
20000
```

In [7]:

```
df = pd.DataFrame({'text':texts, 'title':titles})
df.sample(3)
```

Out[7]:

	text	title
4597	В калифорнийской тюрьме охранники только с пом...	Бунт в тюрьме удалось подавить только огнестре...
17007	Во вторник корпорация Intel признала наличие о...	У Pentium 4 проблемы с BIOS
11877	Начиная с сентября журнал Forbes собирается пу...	В бумажном журнале Forbes будут кликабельные г...

In [8]:

```
from sklearn.model_selection import train_test_split

df_train, df_test = train_test_split(df, test_size=0.1, random_state=1)
```

In [9]:

```
from datasets import Dataset, DatasetDict

ds_data = DatasetDict({
    'train': Dataset.from_pandas(df_train),
    'test': Dataset.from_pandas(df_test)
})

ds_data
```

Out[9]:

```
DatasetDict({
    train: Dataset({
        features: ['text', 'title', '__index_level_0__'],
        num_rows: 18000
    })
    test: Dataset({
        features: ['text', 'title', '__index_level_0__'],
        num_rows: 2000
    })
})
```

In [10]:

```
max_len_text = max(map(lambda txt: len(txt.split()), ds_data['train']['text']))
max_len_tl = max(map(lambda txt: len(txt.split()), ds_data['train']['title']))
max_len_text, max_len_tl
```

Out[10]:

```
(1111, 18)
```

In [11]:

```
max_len_text, max_len_tl = 512, 20
```

Preprocessing the data

In [12]:

```
model_name = "IlyaGusev/rut5_base_sum_gazeta"
```

In [13]:

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained(model_name)
```

In [14]:

```
tokenized_input = tokenizer('привет', padding='max_length', truncation=True, max_length=max
```

In [15]:

```
tokenized_input
```

Out[15]:

```
{'input_ids': tensor([[20842,     1,     0,     0,     0,     0,     0,
0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
```

```
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,
0,
             0,     0]]), 'attention_mask': tensor([[1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0,
        0, 0, 0, 0, 0, 0, 0, 0]])}
```

In [16]:

```
def tokenize(batch):
    tokenized_input = tokenizer(batch['text'], padding='max_length', truncation=True, max_l
    tokenized_label = tokenizer(batch['title'], padding='max_length', truncation=True, max_

    tokenized_input['labels'] = tokenized_label['input_ids']

    return tokenized_input

ds_data = ds_data.map(tokenize, batched=True, batch_size=8)

ds_data.set_format('numpy', columns=['input_ids', 'attention_mask', 'labels'])
```

```
  0%|          | 0/2250 [00:00<?, ?ba/s]

  0%|          | 0/250 [00:00<?, ?ba/s]
```

In [17]:

```
from transformers import DataCollatorForTokenClassification

data_collator = DataCollatorForTokenClassification(tokenizer)
```

Preprocessing the data

In [18]:

```
from transformers import T5ForConditionalGeneration, Trainer, TrainingArguments


model = T5ForConditionalGeneration.from_pretrained(model_name)
```

In [19]:

```
training_args = TrainingArguments(
    "gen_title",
    evaluation_strategy = "epoch",
    learning_rate=1e-5,
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    num_train_epochs= 2,
    remove_unused_columns=True, # Removes useless columns from the dataset
    save_strategy='no',
    report_to='none',
)
```

In [20]:

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=ds_data['train'],
    eval_dataset=ds_data['test'],
    data_collator=data_collator,
    tokenizer=tokenizer,
)
```

In [21]:

```
trainer.train()
```

```
The following columns in the training set don't have a corresponding argu
ment in `T5ForConditionalGeneration.forward` and have been ignored: titl
e, __index_level_0__, text. If title, __index_level_0__, text are not exp
ected by `T5ForConditionalGeneration.forward`,  you can safely ignore thi
s message.
c:\program files\python37\lib\site-packages\transformers\optimization.py:
310: FutureWarning: This implementation of AdamW is deprecated and will b
e removed in a future version. Use the PyTorch implementation torch.opti
m.AdamW instead, or set `no_deprecation_warning=True` to disable this war
ning
  FutureWarning,
***** Running training *****
  Num examples = 18000
  Num Epochs = 2
  Instantaneous batch size per device = 2
  Total train batch size (w. parallel, distributed & accumulation) = 2
  Gradient Accumulation steps = 1
  Total optimization steps = 18000
```

Вывод

Модель обучается, но время на обучение хочет потратить 144 часа, обычно это число у меня еще увеличивается, нету у меня возможности на такое время оставить ноутбук на столе

In [ ]:

```
INX = 100
NMB = 5
print("TEXT: | {}".format(ds_data['test']['text'][INX]))
print("TITLE: | {}".format(ds_daata['test']['title'][INX]))
```

In [ ]:

```
import torch

input_text = ds_data['test']['text'][INX: INX+NMB]

with torch.no_grad():
    tokenized_text = tokenizer(input_text, padding='max_length', truncation=True, max_lengt

    source_ids = tokenized_text['input_ids'].to(dtype = torch.long)
    source_mask = tokenized_text['attention_mask'].to(dtype = torch.long)

    generated_ids = model.generate(
        input_ids = source_ids,
        attention_mask = source_mask,
        max_length=512,
        num_beams=7,
        temperature = 1.3,
        repetition_penalty=1,
        length_penalty=1,
        early_stopping=True,
        no_repeat_ngram_size=2
    )

    for i in range(NMB):
        pred = tokenizer.decode(generated_ids[i], skip_special_tokens=True, clean_up_tokeni
        print("Text: | {}".format(ds_data['test']['text'][INX+i][:1024]))
        print("TITLE: | {}".format(ds_data['test']['title'][INX+i]))
        print("OUTPUT: | {}".format(pred))
        print()
```