In [2]:

```
import torch
from transformers.file_utils import is_tf_available, is_torch_available, is_torch_tpu_avail
from transformers import BertTokenizerFast, BertForSequenceClassification
from transformers import Trainer, TrainingArguments
from datasets import load_dataset, load_metric, Dataset, DatasetDict
from transformers import AutoTokenizer, AutoModel

import pandas as pd

import numpy as np
import random
from sklearn.datasets import fetch_20newsgroups
from sklearn.model_selection import train_test_split
```

Type *Markdown* and LaTeX: α^2

In [3]:

```
def set_seed(seed: int):
    """
    Helper function for reproducible behavior to set the seed in ``random``, ``numpy``, ``t
    installed).

    Args:
        seed (:obj:`int`): The seed to set.
    """
    random.seed(seed)
    np.random.seed(seed)
    if is_torch_available():
        torch.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)
        # ^^ safe to call this function even if cuda is not available
    if is_tf_available():
        import tensorflow as tf

        tf.random.set_seed(seed)

set_seed(1)
```

In [4]:

```
model_name = "bert-base-cased-finetuned-mrpc"
max_length = 512
```

In [5]:

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch

tokenizer = AutoTokenizer.from_pretrained(model_name)
```

```
Downloading tokenizer_config.json:   0%|          | 0.00/29.0 [00:00<?, ?B/
s]

Downloading config.json:   0%|          | 0.00/433 [00:00<?, ?B/s]

Downloading vocab.txt:   0%|          | 0.00/208k [00:00<?, ?B/s]

Downloading tokenizer.json:   0%|          | 0.00/426k [00:00<?, ?B/s]
```

In [6]:

```
corpus = load_dataset('merionum/ru_paraphraser')
```

```
WARNING:datasets.builder:Using custom data configuration merionum--ru_paraph
raser-aad01005423eb70c
WARNING:datasets.builder:Reusing dataset json (C:\Users\voron\.cache\hugging
face\datasets\merionum___json\merionum--ru_paraphraser-aad01005423eb70c\0.0.
0\a3e658c4731e59120d44081ac10bf85dc7e1388126b92338344ce9661907f253)

  0%|          | 0/2 [00:00<?, ?it/s]
```

In [7]:

```
data_train = pd.DataFrame(corpus['train'])
```

In [8]:

```
data_test = pd.DataFrame(corpus['test'])
```

In [9]:

```
data_train = data_train.loc[data_train['class'] != '0']
data_test = data_test.loc[data_test['class'] != '0']
```

In [10]:

```
data_train['class'] = data_train['class'].map({'-1': 0, "1": 1})
data_test['class'] = data_test['class'].map({'-1': 0, "1": 1})
```

In [11]:

```
target_names = {1: 'is_paraphrases',
                0: 'not_paraphrases'}
```

In [12]:

```
data_train['text'] = data_train['text_1'].map(str) + " [SEP] " + data_train['text_2']
data_test['text'] = data_test['text_1'].map(str) + " [SEP] " + data_test['text_2']
```

In [13]:

```
data_train['text'].iloc[1]
```

Out[13]:

```
'Приставы соберут отпечатки пальцев российских должников. [SEP] Приставы сни
мут отпечатки пальцев у злостных неплательщиков.'
```

In [14]:

```
data = data_train[['text', 'class']]
data_test = data_test[['text', 'class']]
```

In [15]:

```
train_corpus = list(data['text'].values)
test_corpus = list(data_test['text'].values)
train_label = list(data['class'].values)
test_label = list(data_test['class'].values)
```

In [16]:

```
list(train_corpus)
```

Out[16]:

```
['Вернувшихся из Сирии россиян волнует вопрос трудоустройства на родине.
[SEP] Самолеты МЧС вывезут россиян из разрушенной Сирии.',
 'Приставы соберут отпечатки пальцев российских должников. [SEP] Приставы
снимут отпечатки пальцев у злостных неплательщиков.',
 'На саратовского дебошира с борта самолета Москва - Хургада заведено дел
о. [SEP] Саратовский дебошир отказывается возвращаться домой из Египта.',
 'Суд Петербурга оставил на потом дело о гибели подростка в полиции. [SE
P] Лондонский Гайд-парк - это не место для митингов, а прежде всего пар
к.',
 'Страны ОПЕК сократили добычу нефти на 1 млн баррелей в день. [SEP] Обам
а продлил полномочия НАСА по сотрудничеству с Россией.',
 'Москвичи смогут забронировать в Интернете место на кладбище. [SEP] В Мо
скве можно будет забронировать место на кладбище через интернет.',
 'Въезд в центр Москвы автомобилям с двигателями Евро-2 не запрещали. [SE
P] Сборная России пробилась в плей-офф чемпионата мира по хоккею с мячо
м.',
 'Северокорейский лидер впервые за 19 лет поздравил граждан с Новым годо
м. [SEP] Лидер КНДР впервые за 19 лет поздравил сограждан с Новым годо
```

In [17]:

```
# tokenize the dataset, truncate when passed `max_length`,
# and pad with 0's when less than `max_length`
train_encodings = tokenizer(train_corpus, truncation=True, padding=True, max_length=max_len
valid_encodings = tokenizer(test_corpus, truncation=True, padding=True, max_length=max_leng
```

In [18]:

```
valid_encodings
```

Out[18]:

```
{'input_ids': [[101, 469, 19692, 17127, 28413, 488, 10286, 488, 19692, 28
406, 28404, 28414, 477, 16948, 28403, 28403, 28404, 10286, 22037, 28394,
28400, 17424, 28394, 10286, 28416, 28404, 28403, 14800, 102, 462, 10286,
20442, 28400, 10286, 28401, 19692, 17127, 28404, 464, 28400, 24625, 1028
6, 28399, 17424, 17424, 490, 16948, 28393, 28400, 10286, 28395, 16948, 28
396, 10286, 20442, 17424, 28400, 488, 10286, 20442, 16948, 28396, 28413,
476, 28413, 28394, 28410, 19692, 28395, 16948, 464, 28382, 28382, 28381,
482, 10286, 462, 16948, 28393, 19692, 28396, 28405, 102, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 107, 451, 16948, 2
8395, 16948, 28400, 28414, 118, 497, 19692, 17127, 28404, 20442, 107, 49
0, 16948, 28399, 10286, 28397, 19692, 28404, 477, 17424, 28396, 19692, 16
948, 28398, 10286, 28402, 17424, 28403, 28414, 492, 28399, 10286, 17127,
28396, 10286, 28400, 28414, 17127, 16948, 28395, 16948, 492, 28402, 1969
2, 28399, 28404, 10286, 28399, 28400, 14800, 107, 465, 10286, 17127, 2839
5, 19692, 17106, 28398, 19692, 20442, 107, 102, 457, 19692, 28407, 28401,
10286, 17127, 482, 10286, 28402, 20442, 19692, 28404, 17424, 28400, 208,
```

In [19]:

```python
class NewsGroupsDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {k: torch.tensor(v[idx]) for k, v in self.encodings.items()}
        item["labels"] = torch.tensor([self.labels[idx]])
        return item

    def __len__(self):
        return len(self.labels)

# convert our tokenized data into a torch Dataset
train_dataset = NewsGroupsDataset(train_encodings, train_label)
valid_dataset = NewsGroupsDataset(valid_encodings, test_label)
```

In [20]:

```python
model = AutoModelForSequenceClassification.from_pretrained(model_name)
```

```
Downloading pytorch_model.bin:   0%|          | 0.00/413M [00:00<?, ?B/s]
```

In [21]:

```
from sklearn.metrics import accuracy_score

def compute_metrics(pred):
  labels = pred.label_ids
  preds = pred.predictions.argmax(-1)
  # calculate accuracy using sklearn's function
  acc = accuracy_score(labels, preds)
  return {
      'accuracy': acc,
  }
```

In [22]:

```
training_args = TrainingArguments(
    output_dir='./results',          # выходной каталог
    num_train_epochs=3,              # общее количество эпох
    per_device_train_batch_size=8,  # размер батча на устройство во время обучения
    per_device_eval_batch_size=20,   # batch размер для оценки
    warmup_steps=500,                # количество шагов разминки для расписания скорости об
    weight_decay=0.01,               # сила снижения веса
    logging_dir='./logs',            # каталог для хранения журналов
    load_best_model_at_end=True,     # загрузите лучшую модель после завершения обучения (м
    # но вы можете указать аргумент `metric_for_best_model` для изменения на точность или д
    logging_steps=400,               # регистрируйте и сохраняйте веса для каждого logging_
    save_steps=400,
    evaluation_strategy="steps",     # оцените каждый `logging_steps`
)
```

```
C:\Users\voron\AppData\Roaming\Python\Python37\site-packages\torch\cuda\__in
it__.py:80: UserWarning: CUDA initialization: The NVIDIA driver on your syst
em is too old (found version 10010). Please update your GPU driver by downlo
ading and installing a new version from the URL: http://www.nvidia.com/Downl
oad/index.aspx (http://www.nvidia.com/Download/index.aspx) Alternatively, go
to: https://pytorch.org (https://pytorch.org) to install a PyTorch version t
hat has been compiled with your version of the CUDA driver. (Triggered inter
nally at  ..\c10\cuda\CUDAFunctions.cpp:112.)
  return torch._C._cuda_getDeviceCount() > 0
```

In [23]:

```
import os
os.environ["WANDB_DISABLED"] = "true"
```

In [24]:

```
trainer = Trainer(
    model=model,                         # созданная модель трансформаторов, подлежащая обу
    args=training_args,                  # обучающие аргументы, определенные выше
    train_dataset=train_dataset,         # обучающий набор данных
    eval_dataset=valid_dataset,          # набор данных для оценки
    compute_metrics=compute_metrics,     # обратный вызов, который вычисляет интересующие п
)
```

In [25]:

```
# train the model
trainer.train()
```

```
C:\Users\voron\AppData\Roaming\Python\Python37\site-packages\transformers\op
timization.py:310: FutureWarning: This implementation of AdamW is deprecated
and will be removed in a future version. Use the PyTorch implementation torc
h.optim.AdamW instead, or set `no_deprecation_warning=True` to disable this
warning
  FutureWarning,
***** Running training *****
  Num examples = 4270
  Num Epochs = 3
  Instantaneous batch size per device = 8
  Total train batch size (w. parallel, distributed & accumulation) = 8
  Gradient Accumulation steps = 1
  Total optimization steps = 1602
```

[1602/1602 7:59:14, Epoch 3/3]

Step	Training Loss	Validation Loss	Accuracy
400	0.497700	0.677475	0.672775
800	0.439100	0.441802	0.828970
1200	0.354900	0.592091	0.841187
1600	0.283300	0.483706	0.840314

```
***** Running Evaluation *****
  Num examples = 1146
  Batch size = 20
Saving model checkpoint to ./results\checkpoint-400
Configuration saved in ./results\checkpoint-400\config.json
Model weights saved in ./results\checkpoint-400\pytorch_model.bin
***** Running Evaluation *****
  Num examples = 1146
  Batch size = 20
Saving model checkpoint to ./results\checkpoint-800
Configuration saved in ./results\checkpoint-800\config.json
Model weights saved in ./results\checkpoint-800\pytorch_model.bin
***** Running Evaluation *****
  Num examples = 1146
  Batch size = 20
Saving model checkpoint to ./results\checkpoint-1200
Configuration saved in ./results\checkpoint-1200\config.json
Model weights saved in ./results\checkpoint-1200\pytorch_model.bin
***** Running Evaluation *****
  Num examples = 1146
  Batch size = 20
Saving model checkpoint to ./results\checkpoint-1600
Configuration saved in ./results\checkpoint-1600\config.json
Model weights saved in ./results\checkpoint-1600\pytorch_model.bin


Training completed. Do not forget to share your model on huggingface.co/mode
ls =)
```

```
Loading best model from ./results\checkpoint-800 (score: 0.441801995038986
2).
```

Out[25]:

```
TrainOutput(global_step=1602, training_loss=0.3933175506253963, metrics={'tr
ain_runtime': 28774.045, 'train_samples_per_second': 0.445, 'train_steps_per
_second': 0.056, 'total_flos': 1270502647457400.0, 'train_loss': 0.393317550
6253963, 'epoch': 3.0})
```

In [26]:

```
model.save_pretrained('sum_bert.bin')
tokenizer.save_pretrained('sum_bert.bin')
```

```
Configuration saved in sum_bert.bin\config.json
Model weights saved in sum_bert.bin\pytorch_model.bin
tokenizer config file saved in sum_bert.bin\tokenizer_config.json
Special tokens file saved in sum_bert.bin\special_tokens_map.json
```

Out[26]:

```
('sum_bert.bin\\tokenizer_config.json',
 'sum_bert.bin\\special_tokens_map.json',
 'sum_bert.bin\\vocab.txt',
 'sum_bert.bin\\added_tokens.json',
 'sum_bert.bin\\tokenizer.json')
```

Применнение

In [27]:

```
PATH = 'sum_bert.bin'
tokenizer = AutoTokenizer.from_pretrained(PATH, local_files_only=True)
model = AutoModelForSequenceClassification.from_pretrained(PATH)
```

```
Didn't find file sum_bert.bin\added_tokens.json. We won't load it.
loading file sum_bert.bin\vocab.txt
loading file sum_bert.bin\tokenizer.json
loading file None
loading file sum_bert.bin\special_tokens_map.json
loading file sum_bert.bin\tokenizer_config.json
loading configuration file sum_bert.bin\config.json
Model config BertConfig {
  "_name_or_path": "sum_bert.bin",
  "architectures": [
    "BertForSequenceClassification"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "problem_type": "single_label_classification",
  "torch_dtype": "float32",
  "transformers_version": "4.21.1",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}

loading weights file sum_bert.bin\pytorch_model.bin
All model checkpoint weights were used when initializing BertForSequenceClas
sification.

All the weights of BertForSequenceClassification were initialized from the m
odel checkpoint at sum_bert.bin.
If your task is similar to the task the model of the checkpoint was trained
on, you can already use BertForSequenceClassification for predictions withou
t further training.
```

In [28]:

```
sequence_0 = data_train['text_1'].iloc[1]
sequence_1 = data_train['text_2'].iloc[1]
sequence_2 = data_train['text_1'].iloc[18]
print(sequence_0)
print(sequence_1)
print(sequence_2)
```

```
Приставы соберут отпечатки пальцев российских должников.
Приставы снимут отпечатки пальцев у злостных неплательщиков.
СК выяснит, был ли подкуп свидетеля по делу Ю.Буданова.
```

In [ ]:

```

```

In [29]:

```
classes = ["не являются парафраза", "являются паравразой"]


paraphrase = tokenizer.encode_plus(sequence_0, sequence_2, return_tensors="pt")
not_paraphrase = tokenizer.encode_plus(sequence_0, sequence_1, return_tensors="pt")

paraphrase_classification_logits = model(**paraphrase)[0]
not_paraphrase_classification_logits = model(**not_paraphrase)[0]

paraphrase_results = torch.softmax(paraphrase_classification_logits, dim=1).tolist()[0]
not_paraphrase_results = torch.softmax(not_paraphrase_classification_logits, dim=1).tolist(

for i in range(len(classes)):
    print(f"{classes[i]}: {round(paraphrase_results[i] * 100)}%")

for i in range(len(classes)):
    print(f"{classes[i]}: {round(not_paraphrase_results[i] * 100)}%")
```

```
не являются парафраза: 96%
являются паравразой: 4%
не являются парафраза: 96%
являются паравразой: 4%
```

In [ ]:

```

```