```
!pip install datasets transformers -q
```

```
import pandas as pd
from tqdm.auto import tqdm

from torch.utils.data import DataLoader
import torch
import torch.nn as nn
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from datasets import load_dataset, DatasetDict
import transformers

from sklearn.metrics import f1_score, accuracy_score
```

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
device
```

```
'cuda'
```

Классификация тональности отзыва на фильм

Решим задачу определения тональности отзыва, датасет возьмём из базы данных онлайн платформы для просмотра Кинопоиск

Подготовка данных

```
data = load_dataset("blinoff/kinopoisk")
data
```

```
/usr/local/lib/python3.12/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
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(
README.md:  1.31k/? [00:00<00:00, 18.2kB/s]
kinopoisk.jsonl: 100%  143M/143M [00:02<00:00, 65.0MB/s]
Generating train split: 100%  36591/36591 [00:04<00:00, 9806.14 examples/s]
DatasetDict({
    train: Dataset({
        features: ['part', 'movie_name', 'review_id', 'author', 'date', 'title', 'grade3', 'grade10', 'content'],
        num_rows: 36591
    })
})
```

```
print(data['train']['content'][100][:100])
print(data['train']['grade3'][100])
```

```
Классика Уолта Диснея — вот те мультфильмы, которые действительно можно считать продуктами этой сту
Bad
```

Разделим исходную выборку

```
dataset = data['train'].train_test_split(test_size=0.2)
test = dataset['test']
train_val = dataset['train'].train_test_split(test_size=0.1)

dataset = DatasetDict({
    'train':train_val['train'],
    'val': train_val['test'],
    'test': test
})

dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['part', 'movie_name', 'review_id', 'author', 'date', 'title', 'grade3', 'grade10', 'content'],
        num_rows: 26344
    })
    val: Dataset({
        features: ['part', 'movie_name', 'review_id', 'author', 'date', 'title', 'grade3', 'grade10', 'content'],
        num_rows: 2928
    })
    test: Dataset({
        features: ['part', 'movie_name', 'review_id', 'author', 'date', 'title', 'grade3', 'grade10', 'content'],
        num_rows: 7319
    })
})
```

В качестве модели для дообучения будет взята rubert-tiny2, загрузим её родной токенизатор

```
model_name = 'cointegrated/rubert-tiny2'
```

```
tokenizer = transformers.AutoTokenizer.from_pretrained(model_name)
```

```
tokenizer_config.json: 100%  401/401 [00:00<00:00, 13.1kB/s]
vocab.txt: 1.08M/? [00:00<00:00, 12.8MB/s]
tokenizer.json: 1.74M/? [00:00<00:00, 15.5MB/s]
special_tokens_map.json: 100%  112/112 [00:00<00:00, 2.52kB/s]
```

```
def preprocess_function(examples):
    model_inputs = tokenizer(examples['content'], max_length=2048, truncation=True)

    labels = {'Good': 2, 'Neutral': 1, 'Bad': 0}
    model_inputs['labels'] = [labels[grade] for grade in examples['grade3']]

    return model_inputs
```

```
tokenized_data = dataset.map(preprocess_function, batched=True)
```

```
Map: 100%  26344/26344 [02:19<00:00, 281.11 examples/s]
Map: 100%  2928/2928 [00:12<00:00, 238.26 examples/s]
Map: 100%  7319/7319 [00:32<00:00, 201.54 examples/s]
```

```
tokenized_data
```

```
DatasetDict({
    train: Dataset({
        features: ['part', 'movie_name', 'review_id', 'author', 'date', 'title', 'grade3', 'grade10', 'content',
'input_ids', 'token_type_ids', 'attention_mask', 'labels'],
        num_rows: 26344
    })
    val: Dataset({
        features: ['part', 'movie_name', 'review_id', 'author', 'date', 'title', 'grade3', 'grade10', 'content',
'input_ids', 'token_type_ids', 'attention_mask', 'labels'],
        num_rows: 2928
    })
    test: Dataset({
        features: ['part', 'movie_name', 'review_id', 'author', 'date', 'title', 'grade3', 'grade10', 'content',
'input_ids', 'token_type_ids', 'attention_mask', 'labels'],
        num_rows: 7319
    })
})
```

```
model = transformers.AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=3)
```

```
config.json: 100%  693/693 [00:00<00:00, 9.19kB/s]
model.safetensors: 100%  118M/118M [00:03<00:00, 35.2MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at cointegrated/rubert-tiny2 ar
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
model
```

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(83828, 312, padding_idx=0)
      (position_embeddings): Embedding(2048, 312)
```

```
      (token_type_embeddings): Embedding(2, 312)
      (LayerNorm): LayerNorm((312,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-2): 3 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=312, out_features=312, bias=True)
              (key): Linear(in_features=312, out_features=312, bias=True)
              (value): Linear(in_features=312, out_features=312, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=312, out_features=312, bias=True)
              (LayerNorm): LayerNorm((312,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=312, out_features=600, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=600, out_features=312, bias=True)
            (LayerNorm): LayerNorm((312,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=312, out_features=312, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=312, out_features=3, bias=True)
)
```

```
data_collator = transformers.DataCollatorWithPadding(tokenizer=tokenizer)
```

В течении обучения будем вычислять дополнительные метрики для отчётности после каждой эпохи

```
def compute_metrics(eval_pred):
  predict, labels = eval_pred
  predict = np.argmax(predict, axis=1)

  accuracy = accuracy_score(labels, predict)
  f1_weighted = f1_score(labels, predict, average='weighted')
  f1_macro = f1_score(labels, predict, average='macro')

  return {'accuracy': accuracy, 'f1_weighted': f1_weighted, 'f1_macro': f1_macro}
```

```
training_args = transformers.TrainingArguments(
        output_dir="./results_version_0.1",
        eval_strategy='epoch',
        save_strategy='epoch',
        learning_rate=2e-5,
        per_device_train_batch_size=32,
        per_device_eval_batch_size=32,
        weight_decay=0.1,
        save_total_limit=5,
        num_train_epochs=5,
        report_to = 'none',
        logging_strategy='epoch',
        load_best_model_at_end=True,
        metric_for_best_model='f1_weighted',
        greater_is_better=True,
    )
```

```
trainer = transformers.Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_data["train"],
        eval_dataset=tokenized_data["test"],
        processing_class=tokenizer,
        data_collator=data_collator,
        compute_metrics=compute_metrics
    )
```

```
trainer.train()
```

[4120/4120 1:19:46, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Accuracy	F1 Weighted	F1 Macro
1	0.491100	0.378038	0.851756	0.825835	0.642280
2	0.348400	0.370536	0.852439	0.844625	0.695834
3	0.317800	0.361697	0.861730	0.850172	0.704325
4	0.298500	0.369257	0.861866	0.851905	0.708788
5	0.282400	0.371240	0.860773	0.852046	0.710343

```
TrainOutput(global_step=4120, training_loss=0.3476305952349913, metrics={'train_runtime': 4788.5857,
'train_samples_per_second': 27.507, 'train_steps_per_second': 0.86, 'total_flos': 2363976822066336.0, 'train_loss':
```

```
model_path = "/content/drive/MyDrive/rubert"
```

```
model = transformers.AutoModelForSequenceClassification.from_pretrained(model_path)
tokenizer = transformers.AutoTokenizer.from_pretrained(model_path)
```

```
y_pred = trainer.predict(tokenized_data['test'])
```

```
y_true = y_pred.label_ids
y_pred_ = np.argmax(y_pred.predictions, axis=1)
```

```
print(accuracy_score(y_true, y_pred_))
f1_score(y_true, y_pred_, average='weighted')
```

```
0.8891925126383385
0.8819655063830788
```

Сохраним модель для последующих эксперемнтов

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

```
Mounted at /content/drive
```

```
trainer.save_model("/content/drive/MyDrive/rubert")
```

```
tokenizer.save_pretrained("/content/drive/MyDrive/rubert")
```

```
('/content/drive/MyDrive/my_best_model/tokenizer_config.json',
 '/content/drive/MyDrive/my_best_model/special_tokens_map.json',
 '/content/drive/MyDrive/my_best_model/vocab.txt',
 '/content/drive/MyDrive/my_best_model/added_tokens.json',
 '/content/drive/MyDrive/my_best_model/tokenizer.json')
```