

# Named Entity Recognition. DistilBERT

. . ИУ5-22М

```
In [1]: from datasets import load_dataset
        from transformers import DataCollatorForTokenClassification
        from transformers import AutoTokenizer, AutoModelForTokenClassification
        from transformers import TrainingArguments, Trainer

        import numpy as np
        import evaluate
```

## Загрузка набора данных

Для обучения будем использовать "русскую" часть [WikiNEuRal](#)

```
In [2]: # Загрузка датасета
        dataset = load_dataset("Babelscape/wikineural")
```

```
In [3]: # Пример строки из датасета
        dataset["train_ru"][0]
```

```
Out[3]: {'tokens': ['Детство',
                    'провёл',
                    'в',
                    'Надьсомбате',
                    ',',
                    'с',
                    '1860',
                    'г',
                    '.'],
         'ner_tags': [0, 0, 0, 5, 0, 0, 0, 0, 0],
         'lang': 'ru'}
```

## Предобработка данных

```
In [4]: # Загрузка токенизатора
        tokenizer = AutoTokenizer.from_pretrained(
            "distilbert/distilbert-base-multilingual-cased"
        )
```

```
In [5]: # Пример работы токенизатора
        example = dataset["train_ru"][0]
        tokenized_input = tokenizer(example["tokens"], is_split_into_words=True)
        tokens = tokenizer.convert_ids_to_tokens(tokenized_input["input_ids"])
        tokens
```

```
Out[5]: ['[CLS]',
        'Де',
        '##тство',
        'провёл',
        'в',
        'Над',
        '##ь',
        '##сом',
        '##бат',
        '##е',
        ',',
        'с',
        '1860',
        'Г',
        '.',
        '[SEP]']
```

```
In [6]: def tokenize_and_align_labels(examples):
        """Корректировка токенизации
        Parameters
        -----
        examples
            Входное предложение
        Returns
        -----
            tokenized_inputs
            Токенизированный вход
        """
        tokenized_inputs = tokenizer(
            examples["tokens"], truncation=True, is_split_into_words=True
        )

        labels = []
        for i, label in enumerate(examples[f"ner_tags"]):
            word_ids = tokenized_inputs.word_ids(batch_index=i)  # Токенизация
            previous_word_idx = None
            label_ids = []
            for word_idx in word_ids:  # Установка значения спец. токенов -100
                if word_idx is None:
                    label_ids.append(-100)
                elif (
                    word_idx != previous_word_idx
                ):  # Применяем метку только к первому слову в предложении при нескольк
                    label_ids.append(label[word_idx])
                else:
                    label_ids.append(-100)
                    previous_word_idx = word_idx
            labels.append(label_ids)

        tokenized_inputs["labels"] = labels
        return tokenized_inputs
```

```
In [7]: # Применение токенизатора к датасету
        tokenized_dataset = dataset.map(tokenize_and_align_labels, batched=True)
```

Map: 0% | 0/11069 [00:00<?, ? examples/s]

```
In [8]: # Загрузка DataCollator
data_collator = DataCollatorForTokenClassification(tokenizer=tokenizer)
```

## Обучение модели

### Метрики качества

```
In [9]: sequeval = evaluate.load("sequeval")
```

```
In [12]: def compute_metrics(p):
    """Функция для расчёта метрик
    Parameters
    -----
    p
        Предсказание
    Returns
    -----
    metrics
        Метрики качества
    """
    predictions, labels = p
    predictions = np.argmax(predictions, axis=2)
    label_list = [
        "O",
        "B-PER",
        "I-PER",
        "B-ORG",
        "I-ORG",
        "B-LOC",
        "I-LOC",
        "B-MISC",
        "I-MISC",
    ]

    true_predictions = [
        [label_list[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]
    true_labels = [
        [label_list[l] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]

    results = sequeval.compute(predictions=true_predictions, references=true_labels)
    return {
        "precision": results["overall_precision"],
        "recall": results["overall_recall"],
        "f1": results["overall_f1"],
        "accuracy": results["overall_accuracy"],
    }
```

## Загрузка и обучение модели

Для обучения будем использовать [мультязычную версию distilBERT](#)

```
In [10]: id2label = {
          0: "O",
          1: "B-PER",
          2: "I-PER",
          3: "B-ORG",
          4: "I-ORG",
          5: "B-LOC",
          6: "I-LOC",
          7: "B-MISC",
          8: "I-MISC",
        }
        label2id = {
          "O": 0,
          "B-PER": 1,
          "I-PER": 2,
          "B-ORG": 3,
          "I-ORG": 4,
          "B-LOC": 5,
          "I-LOC": 6,
          "B-MISC": 7,
          "I-MISC": 8,
        }
```


```
In [11]: # Загрузка модели
        model = AutoModelForTokenClassification.from_pretrained(
            "distilbert/distilbert-base-multilingual-cased",
            num_labels=9,
            id2label=id2label,
            label2id=label2id,
        )
```

Some weights of DistilBertForTokenClassification were not initialized from the model checkpoint at distilbert/distilbert-base-multilingual-cased and are newly initialize d: ['classifier.bias', 'classifier.weight']  
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

```
In [13]: # Аргументы для обучения
training_args = TrainingArguments(
    output_dir="distilBERT",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    push_to_hub=False,
)
```

```
In [14]: # Описание тренера
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train_ru"],
    eval_dataset=tokenized_dataset["val_ru"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
```

```
In [15]: # Обучение модели
trainer.train()
```

 [17310/17310 34:58, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Precision	Recall	F1	Accuracy
1	0.047400	0.042902	0.882115	0.899506	0.890726	0.985407
2	0.029300	0.041630	0.893203	0.912136	0.902570	0.986338
3	0.018500	0.043006	0.902096	0.912301	0.907170	0.987169

```
Out[15]: TrainOutput(global_step=17310, training_loss=0.03671303313292223, metrics={'train_runtime': 2098.6184, 'train_samples_per_second': 131.973, 'train_steps_per_second': 8.248, 'total_flos': 5941936302049440.0, 'train_loss': 0.03671303313292223, 'epoch': 3.0})
```

```
In [16]: # Качество лучшей модели
trainer.evaluate()
```

 [722/722 00:16]

```
Out[16]: {'eval_loss': 0.0416296049952507,  
          'eval_precision': 0.8932028393202839,  
          'eval_recall': 0.9121361889071938,  
          'eval_f1': 0.9025702331141662,  
          'eval_accuracy': 0.9863384575648139,  
          'eval_runtime': 19.1895,  
          'eval_samples_per_second': 601.372,  
          'eval_steps_per_second': 37.625,  
          'epoch': 3.0}
```

```
In [17]: # Сохранение модели  
trainer.save_model("distilBERT/distilBERT_best_model/")
```