Project Technical Report

Problem statement

Description

We are creating a model that will help the procurement department extract the necessary piece of text from the document in order to create a request form. The specific fragment of text to be extracted depends on the section of the form corresponding to the document. Each document contains one of two sections of the form, from which pieces of text need to be extracted:

- `обеспечение исполнения контракта` (contract fulfillment guarantee)
- `обеспечение гарантийных обязательств` (warranty obligations guarantee)

Data format


```
[
  {
    "id": 809436509,
    "text": "Извещение о проведении открытого конкурса в электронной форме для закупки №0328300032822000806 Общая информация Номер изв
    "label": "обеспечение исполнения контракта",
    "extracted_part": {
      "text": [
        "Размер обеспечения исполнения контракта 6593.25 Российский рубль"
      ],
      "answer_start": [
        1279
      ],
      "answer_end": [
        1343
      ]
    }
  },
  {
      "id": 885717075,
      "text": "Приложение №3 к извещению о проведении электронного аукциона ПРОЕКТ ГОСУДАРСТВЕННОГО КОНТРАКТА № _____ Идентифика
      "label": "обеспечение гарантийных обязательств",
      "extracted_part": {
        "text": [
          "Обеспечение гарантийных обязательств установлено в размере 5 % от начальной (максимальной) цены Контракта, что составляет 1
        ],
        "answer_start": [
          1187
        ],
        "answer_end": [
          1349
        ]
      }
    },
  // ...
]
```


Model requirements

A model taking as input the `текст документа` (document text) and the `наименование одного из двух пунктов` (name of one of the two sections), should return the `соответствующий кусочек текста из текста документа` (corresponding piece of text from the document text).

A model should be able to extract the necessary fragment of text from the document text based on the pair of `текст документа` (document text) and `пункт анкеты` (section of the form).

Evaluation

The `Accuracy` metric will be used to evaluate the final solution: the proportion of observations in which the text fragment extracted by the model fully corresponds to the required fragment.

Data collection and preprocessing

Origin of the dataset

The dataset is introduced by “SKB Kontur”. It consist of a collection of Russian public procurement documents.

Label adjustment

Some extracted parts contain an amount of money anonymised using underscores. These underscores are usually in the end of extracted parts and may or may not include all the consecutive symbols. To make the data less ambiguous, we decided to treat all the consecutive underscores as one token (even though most tokenisers treat one underscore as one token).

```
Original text
... от цены Контракта и составляет _____ (_____). ...

Extracted part before preprocessing
... от цены Контракта и составляет _____ (_____

Extracted part after preprocessing
... от цены Контракта и составляет _____ (_____).
```

Input truncation

The document size is usually too big to fit into the model. We analysed the dataset and came to the conclusion that the target is most likely to be in the middle of the document. To solve this problem, we removed irrelevant sentences from the beginning and the end of the document using sentence embeddings.

Punctuation restoration

The documents from the dataset often lack punctuation. Many documents missed the periods between sentences which made it harder to split texts into sentences. We decided to place periods using regex which took into account words that started with capital letters.

Review of methods and models

Regex

Since the inputs are documents with the similar structure, it would be possible to write a regex to solve the problem. However, it is challenging to write a regex for complex patterns and maintain it. Moreover, good regex accuracy requires thorough dataset analysis.

Named entity recognition

In NER, algorithms typically use a combination of features, such as the words in the sentence, their part-of-speech tags, and their dependency relations, to make predictions about the presence and boundaries of named entities.

What makes it hard to use NER to solve our problem:

- NER struggles with longer entities due to their complexity and multiple possible interpretations.
- Longer entities require more context to be correctly identified within a sentence or document.
- NER algorithms need to consider both the individual words in the entity and their context to correctly identify longer entities.

All this makes NER inaccurate and computationally inefficient.

Spancat

The SpanCategorizer is a spaCy component that answers the NLP community's need to have structured annotation for a wide variety of labeled spans, including long phrases, non-named entities, or overlapping annotations.

Reasons why Spancat seemed appealing to us:

- Spancat offers a variety of suggester functions for information extraction tasks.
- Spancat can be seamlessly integrated into existing spaCy pipelines
- The SpanFinder component identifies span boundaries by tagging potential start and end tokens. It's an ML approach to suggest fewer but more precise span candidates than the ngram suggester. This reduces a number of suggester candidates, which generally solves a problem.

What makes it not the best option:

- Spancat requires high computational power(mostly in terms of memory and time) to train a model for extraction of spans with a length over 40 tokens
- SpanFinder does not significantly reduce a number of suggested candidate in case of long extracting spans

Mentioned challenges make Spancat unsuitable for addressing the problem at hand

Question Answering

In QA, task that involves automatically generating an answer to a user's question based on a given context. In our setup, it is a fact-based QA which aim to find an answer to label passed as a question within specific document context. These question will have a single correct answer that can be found within a document. Therefore, it perfectly suits in our problem and we should only do the proper preprocessing of the data.

What can decrease performance of QA:

- Poor annotations, meaning that extraction parts have inconsistency, while the data itself is not clean and have problems with punctuation
- Small amount of training samples

Architecture and implementation

As a solution, we choosed combination of `ruBert-large` transformer (by ai-forever) and fact-based Question Answering as the downstream task. The architecture can be broken down into the following steps:

1. The input document is preprocessed by tokenizing it into subword units using the Byte-Pair Encoding algorithm, which is used by the ruBert model. The tokens are then converted into numerical embeddings using the ruBert model's embedding layer.
2. The input document embeddings are then fed into the ruBert transformer model, which consists of a stack of multiple self-attention layers. Each self-attention layer in the transformer model learns to attend to different parts of the input sequence and extract relevant features. The output of the last self-attention layer is a set of contextualized embeddings that capture the meaning of each token in the input sequence.
3. The input question is also preprocessed and tokenized using BPE, and the resulting tokens are fed into the embedding layer of the ruBert model to obtain question embeddings.
4. The question embeddings are then used to compute attention scores over the contextualized embeddings of the input document. The attention scores measure the relevance of each token in the input document to the question.
5. The tokens with the highest attention scores are selected as the answer span, and the corresponding text is extracted from the input document.
6. The extracted answer span is then post-processed to return the final answer start and end index.
7. Additionally, when model predicts very short extracted part (less than or equal 30 chars) we assume it for empty extraction.

The implementation is done using Python and libraries PyTorch, Hugging Face transformers, datasets. Can be found in repository in the `qa` folder.

> 📌 Models were trained on **NVIDIA GeForce RTX 3080 12 GB**

Evaluation and conclusion

Model	QA	NER
`tok2vec`		64.15%
`cointegrated/rubert-tiny2`	55.35%	67.60%
`M-CLIP/M-BERT-Distil-40`	72.33%	
`base-base-multilingual-cased`		73.33%
`DeepPavlov/rubert-base-cased (baseline)`	75.80%	
`distilbert-base-multilingual-cased`	78.62%	
`ai-forever/sbert_large_mt_nlu_ru`	83.65%	
`ai-forever/ruBert-large`	**84.91%**	

QA models generally outperform NER models in the given problem. Among the QA models, the highest performing model is the `ai-forever/ruBert-large` with an accuracy of 84.91%. On the other hand, among the NER models, `DeepPavlov/rubert-base-cased` achieved the highest accuracy of 75.80%.

Regarding evaluation of Spancat, training was possible only with a limiting of maximum span length to 15 tokens, although the maximum length in dataset is 71. This led to an accuracy of around 55%, which is a significantly low considering the resources spent on learning.

Each team member contribution

Vladimir Makharev

Data collection, data exploration, review of methods, fine-tuning of NER, spancat, QA, model evaluation

Danil Andreev

Data exploration, review of methods, document core extraction, spancat fine-tuning

Mikhail Fedorov

Data exploration, data preprocessing, fine-tuning of models, overview of Spancat

Link to GitHub

https://github.com/kilimanj4r0/nlp-ie-task (write to @sm1rk in Telegram to get access)