FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Adam Szabó

Low-resource Text Classification

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	RNDr. Milan Straka, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2021

I would like to thank the supervisor of this thesis RNDr. Milan Straka, Ph.D. for his valuable advice, patience and professional assistance in the elaboration of this thesis. I would also like to thank Profinit EU s.r.o. for allowing me to work on this work in their Big Data department and the head of the department, Ing. Marek Sušický for his expertise. Furthermore, thanks also go to Michal Bláha for providing the source data needed to create the dataset of contracts. Last but not least, I thank my parents for their support during my studies.

Title: Low-resource Text Classification

Author: Bc. Adam Szabó

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Milan Straka, Ph.D., Institute of Formal and Applied Linguistics

Abstract: The aim of the thesis is to evaluate Czech text classification tasks in the low-resource settings. We introduce three datasets, two of which were publicly available and one was created partly by us. This dataset is based on contracts provided by the web platform Hlídač Státu. It has most of the data annotated automatically and only a small part manually. Its distinctive feature is that it contains long contracts in the Czech language. We achieve outstanding results with the proposed model on publicly available datasets, which confirms the sufficient performance of our model. In addition, we performed experimental measurements of noisy data and of various amounts of data needed to train the model on these publicly available datasets. On the contracts dataset, we focused on selecting the right part of each contract and we studied with which part we can get the best result. We have found that for a dataset that contains some systematic errors due to automatic annotation, it is more advantageous to use a shorter but more relevant part of the contract for classification than to take a longer text from the contract and rely on BERT to learn correctly.

Keywords: text classification, low-resource, BERT

Contents

Introduction

Nowadays, text classification is an increasingly important part of various systems. Consider for example spam detection or the increasingly widespread social networks and their comments, which are a direct example of sentiment analysis task. Until recently, similar problems were being solved with methods such as Random Forest, Support Vector Machines (SVM) [Cortes and Vapnik, 1995] or Maximum Entropy [Harremoës and Topsøe, 2001]. Research about natural language processing (NLP) tasks have become even more widespread after the success of neural networks and has led to the use of deep learning or word embeddings-based methods. However, in 2018 a new model of language representation called BERT (Bidirectional Encoder Representations from Transformers) [Devlin et al., 2018] was presented and has moved almost all the results obtained so far for NLP to a higher level. Its main difference from the older models is that it was designed to pre-train deep bidirectional representations from an unlabeled text by considering both the left and right contexts in all layers [Devlin et al., 2018]. BERT is very easily fine-tuned with only one additional output layer to obtain a model that overcomes the mentioned older models for NLP tasks. That is why our goal in this work is to choose the right variant of BERT and fine-tune it well enough so that we can solve various classification tasks in the Czech language in the low-resource setting.

Despite the good results of recent years, computers are still struggling with different language aspects in NLP tasks which are much clearer for humans. These are, for example, pragmatics, which is formally difficult to characterize, or linguistic irony, which sometimes makes a problem even for people. Most of the best results are obtained in popular languages that have very large text corpora, such as English. There are only a few languages like this, and the rest of the languages need tools or other resources to overcome this barrier to get good results in NLP tasks. They are known as low-resource languages and Czech is one of them.

Our Contribution

In our work, we present the BERT model and the Transformers architecture due to their use in our model. We use publicly available datasets to determine the actual performance of our model. The first is a Facebook dataset [Habernal et al., 2013a], on which we solve the sentiment analysis task and achieve very good results. Furthermore, we were curious about how powerful our model is, and we started experimented with amount of training data needed to achieve similarly good results. We found that to overcome the results on this dataset obtained with older methods such as SVM or Maximum Entropy, we only need to take a few percents of the training set. To make the task more difficult for our model, we also replaced certain amount of training and development data by random incorrect labels.

The second public dataset is the Czech Text Document Corpus, on which we solve multi-label classification in contrary to the previous dataset solving single-label classification. The achieved result is to our knowledge the best that has been measured on this dataset so far. For this reason, we also experimented with the amount of training data.

These experiments encouraged us to create a third dataset. It is based on a real problem of the classification of contracts into 105 categories, of which 22 are the main ones. We obtained the contracts from the Hlídač Státu[1] web platform, which automatically classifies these contracts using a keyword method. However, for this reason it contains only noisy annotations, which contains systematic classification errors and not random. We further managed to acquire a small number of manually classified contracts, which form the test set. Since these contracts are very long documents, mostly in the Czech language, and there is no similar dataset, we decided to publish it for further experiments. We added the main categories to the individual contracts in dataset preparation according to the relevance of the Hlídač Státu, and the ranges of the text where there are the most keywords. A list of keywords that Hlídač Státu uses for classification was also provided to us. We focused on the selection of the amount and the correct part of the contract, which we present to our model for classification. We have found that it is not worth using a large amount of text from the contracts and relying on pre-training the model in this case. The better option is to use a smaller part of the contract like its subject or the first few hundred tokens of the contract, which reduces the propagation of systematic errors in our model.

[1] `https://www.hlidacstatu.cz`

1. Theoretical Background

In this chapter we introduce the pre-trained model, which we use, and its individual components. Firstly, we describe the Transformer architecture. It is based on the mechanism of attention, which completely omits recurrence and convolutions. We describe the essential parts and their operating principle according to the paper Vaswani et al. [2017]. Secondly, we add more details about the principle of the operation of Bidirectional Encoder Representations from Transformers (BERT). The presented principles are mainly based on the paper Devlin et al. [2018]. If you are familiar with this model, you can safely skip the chapter.

1.1 Transformer

The Transformer architecture is an alternative to a Recurrent Neural Network (RNN) and also to convolution approaches in sequence representation. For illustration, we have a sequence and either we want to perform sequence-to-sequence operations to generate another sequence, or we want to represent the elements of the sequence, or represent the sequence as a whole. Sequential processing of its elements, as performed by recurrent neural networks, might be too restrictive. Instead of that, we want to be able to combine sequence elements independently on their distance. Such processing is allowed in the Transformer architecture, initially proposed for neural machine translation in Vaswani et al. [2017]. The foundations are built on the encoder-decoder architecture.

The Figure 1.1 is the main figure in the original paper about Transformer architecture. Generally, it works very similarly to what originally sequence-to-sequence architecture did. In the original sequence-to-sequence architecture, we started with the input sequence. It was processed by the encoder and the encoder was repeated application of RNN cell. Then we get the decoder in the RNN architectures. The decoder generated the output sequence which was also passed on the input but shifted by one. In the decoder we also used RNN cells as the main architecture and finally, we also used the attention mechanism. The attention mechanism was used to combine the state of the decoder and the elements from the encoder. Generally, the Transformer architecture keeps the same overview settings, but it replaces the RNNs with so-called self-attention.

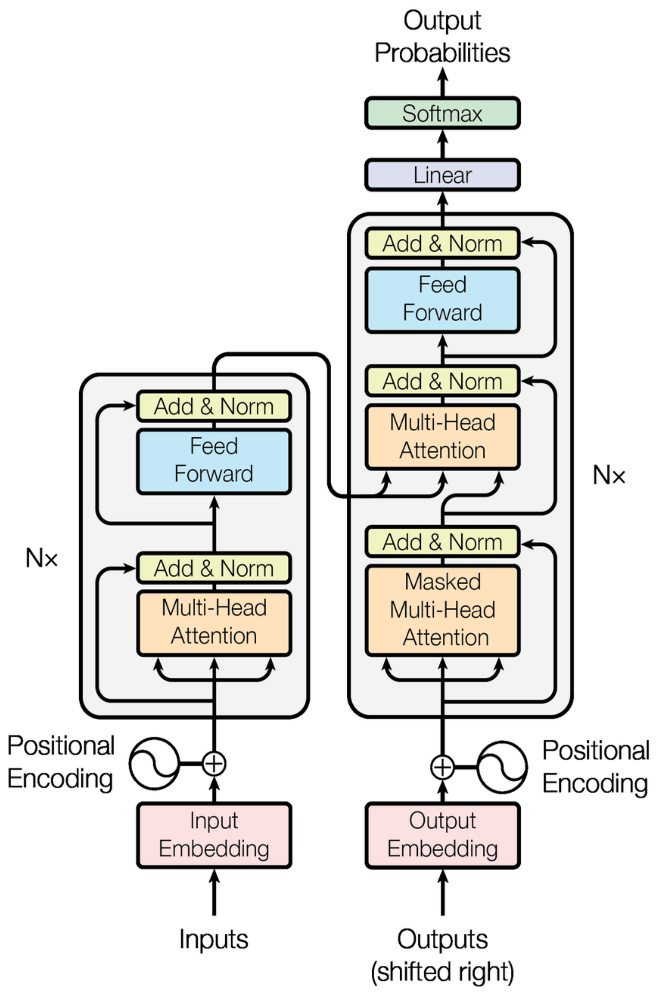


Figure 1.1: Architecture of the Transformer, taken from paper Vaswani et al. [2017].

In the encoder, the self-attention is followed by a local Feed Forward Network. In the decoder we use self-attention again, followed by a so-called encoder-decoder attention, which is like the original attention in machine translation, combining encoder-decoder representation. They are illustrated in Figure 1.2. As well as the encoder, the outputs are fed to a feed-forward neural network. The final linear layer creates the words from a vector of floats produced by the decoder stack, followed by a softmax layer. The self-attention must attend only to earlier positions in the output sequence during decoding. This is achieved by masking future positions, zeroing their weights out, which is usually implemented by setting them to $-\infty$ before softmax calculation. We use the self-attention to combine the value in the sequence and the local feed-forward neural network we use to perform non-linearities on the individual words.

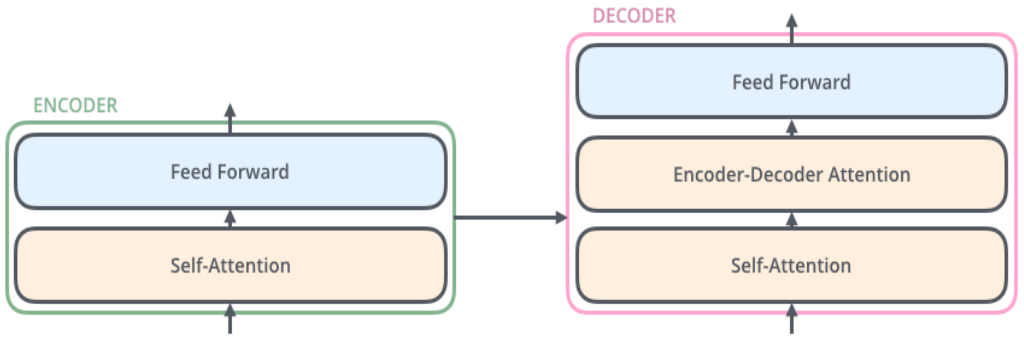


Figure 1.2: Transformer decoder, taken from `http://jalammar.github.io/images/t/Transformer_decoder.png`.

1.1.1 Self-Attention

The main goal of self-attention is to allow combining each input word with any other word in the sequence. We do this by allowing each word to provide three signs. One sign is the query sign and it represents what a word is searching for. The other sign, the key sign, indicates what a word is offering. These keys and queries match together to indicate how much a specific word wants to copy the information from another. Finally, each word also provides a value sign, which contain the actual information to be copied (the value that was advertised via key sign). Concretely, assume that we have the sequence of n words and we represent those using the input representation matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, where d is the dimension of elements of the input words. We start by computing the so-called queries $\mathbf{Q} \in \mathbb{R}^{n \times d_k}$, keys $\mathbf{K} \in \mathbb{R}^{n \times d_k}$ and values $\mathbf{V} \in \mathbb{R}^{n \times d_v}$ from the input word representation $\mathbf{X}$ using a linear transformation as

$$\mathbf{Q} = \mathbf{W}^Q \cdot \mathbf{X} \tag{1.1}$$
$$\mathbf{K} = \mathbf{W}^K \cdot \mathbf{X} \tag{1.2}$$
$$\mathbf{V} = \mathbf{W}^V \cdot \mathbf{X}. \tag{1.3}$$

Then the attention, sometimes called as scaled dot-product attention, can be computed as

$$\text{Attention}\left(\mathbf{Q}, \mathbf{K}, \mathbf{V}\right) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}, \tag{1.4}$$

where d is the dimension of the key vectors.

1.1.2 Multihead Attention

In the self-attention, we choose a single distribution according to which we copy information. However, performing just one decision where we want to copy the information from may not be enough. Therefore, we usually extend the self-attention to a so-called multihead attention, which is illustrated in Figure 1.3. Instead of using one huge attention, we split queries keys and values into several groups, compute the attention in each of the groups separately, and concatenate the results, which are finally passed through a linear transformation.

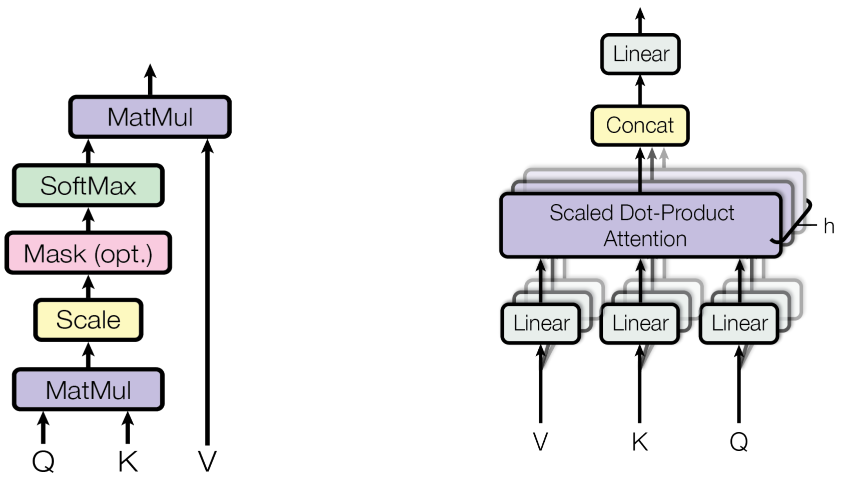


Figure 1.3: Scaled Dot-Product Attention (left), Multi-Head Attention (right) consists of several attention layers running in parallel. Taken from paper Vaswani et al. [2017].

1.1.3 Feed Forward Networks

The self-attention is complemented with a Feed Forward Network layer, which is fully composed of a connected ReLU layer with four times as many hidden units as inputs, followed by another fully connected layer without activation.

$$\mathrm{FFN}(\mathbf{x}) = \mathrm{ReLU}(\mathbf{x}W_1 + b_1)W_2 + b_2. \tag{1.5}$$

1.1.4 Positional Encoding

The proposed self-attention layer does not contain any recurrence and any convolution, and therefore it completely ignores word order. Still, the word positions can naturally be crucial. To make use the order of the sequence in the model, we need to encode positional information, which was implicit in RNNs. To this end, Positional Encodings PEs are added to the input embeddings at the beginning of the encoder and decoder stacks. We can obtain PE by using learned embeddings for every position. However, Vaswani et al. [2017] shows how the PEs can be constructed explicitly using the sine and the cosine functions of different frequencies:

$$\mathrm{PE}_{(pos,2i)} = \sin(pos/10\,000^{2i/d}) \tag{1.6}$$

$$\mathrm{PE}_{(pos,2+i)} = \cos(pos/10\,000^{2i/d}), \tag{1.7}$$

where *pos* is the position and i is the dimension. The PEs have the same dimension d as the embeddings. Note, that this choice of functions allow the model to attend to relative positions, since for any fixed k, PE_{pos+k} is a linear function of PE_{pos}.

1.1.5 Training Procedure

We start by describing *Regularization.* The dropout is applied everywhere just before adding the residual connection. Furthermore, the network is regularized by *Label smoothing.* For both dropout and *Label smoothing*, the default weight of 0.1 is used. Note, that using the usual rate of 0.5 for dropout would considerably decrease the model capacity and hurt performance.

Because the Transformer architecture does not use any recurrent computation, it allows to train the decoder for all words at the same time. To that end, we use teacher forcing (the gold labels are used as predictions in the decoder, which is commonly used in encoder-decoder architectures), and more importantly also *masked attention*, which prevents the self-attention and encoder-decoder attention to attend to later words, which will not be available during prediction. However, the inference is still sequential.

Training is performed using *Adam* optimizer [Kingma and Ba, 2014] with a slightly smaller value for the second momentum decay ($\beta_2 = 0.98$, smaller than the default value of 0.999). The learning rate during training decreases proportionally to the inverse square root of the step number. Furthermore, during the first warmup steps updates, the learning rate is increasing linearly from zero to its target value:

$$\frac{1}{\sqrt{d_{model}}} \min\left(\frac{1}{\sqrt{step\_num}}, \frac{step\_num}{warmup\_steps} \cdot \frac{1}{\sqrt{warmup\_steps}}\right). \tag{1.8}$$

In the original paper Vaswani et al. [2017], 4000 warmup steps were proposed.

Generally, Transformer provide more powerful sequence-to-sequence architecture and also sequence element representation architecture than RNNs, but usually requires substantially more data.

1.2 BERT

BERT is a shortcut for Bidirectional Encoder Representations from Transformers. It is a type of language model developed and released by Google in the article Devlin et al. [2018]. BERT model is based on the Transformer architecture, on its encoder to be exact. Nowadays, the pre-training approaches are dominating for generating textual embeddings and word embeddings. The BERT model computes contextualized representations in a bidirectional way. The concept of bidirectionality is the main distinguishing feature compared to the previous model OpenAI GPT [Radford et al., 2018]. It is important but it makes training difficult.

The BERT model's input are two so-called sentences. When we say sentences, we do not mean real sentences, but segments of text. The maximum number of subwords is 512, so these sentences can easily contain hundreds of words each. The sentences are generally short and this amount of subwords is sufficient. Representing input words as single tokens has several drawbacks, most importantly that many words will not be representable. Instead, BERT represents each word using possibly several subwords, which allows each word to be represented. Specifically, BERT uses the WordPiece approach [Wu et al., 2016] for constructing the subword dictionary. Furthermore, when we mention tokens in the context of BERT, we mean subwords.

The first token is a special CLS (classification) token, which is added to the beginning of sentences when they are given to BERT and a SEP (separation) token ends every sentence. Additionally, a trainable embedding indicates if a token belongs to sentence A (inclusively up to its SEP token) or sentence B, it is illustrated on Figure 1.4. When BERT is applied on a down-stream task, it is pre-trained on raw text and then fine-tuned on the supervised task data.

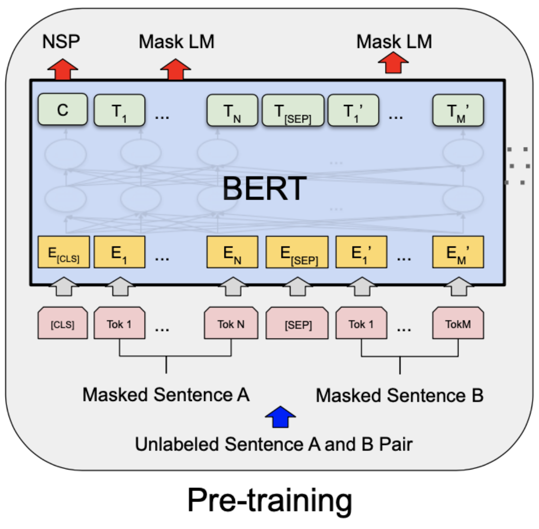


Figure 1.4: Pre-training procedures for BERT. Taken from Devlin et al. [2018].

1.2.1 Pre-Training

The pre-training itself is performed on English on a union of two corpora. The first is English BookCorpus containing 800 million words and the second one is English Wikipedia with 2.5 billion words. Totally is just slightly more than 3 billion words. The WordPieces vocabulary with 30 thousands word pieces is used. During pre-training, quite large batches are used. Each batch contains 256 sequences and each sequence contains two sentences with a total length of 512 subwords. All together there are 128 thousands tokens per batch which is a really large value. Adam is usually used with learning rate 1e-4 and with linear learning rate warmup for the first 10 thousands steps but without decay. The network is trained for 1 million updates and weight decay of 0.01 is used. Similar to the Transformer architecture dropout of 10% is used on all layers, but the Gaussian Error Linear Units (GELU) activation is used instead of Rectified Linear Unit (ReLU). Because the self-attention has quadratic complexity, longer sequences are more expensive than shorter ones. Therefore, in order to the training be more efficient, first 90% of the pre-training is performed on sequences of length 128 and only the last 10% use sequences of length 512.

GELU

Following OpenAI GPT [Radford et al., 2018], the BERT model uses GELU activation. To describe the GELU activation, first consider the ReLU activation function: if the input is positive, it is multiplied by one, and if the input is negative, it is multiplied by zero. Dropout actually follows a similar scheme, either multiplying the input by zero or one, but independently on the input value, in a stochastic way.

Both these approaches are merged in GELU, where the input value $\boldsymbol{x}$ is multiplied by $\boldsymbol{m}\sim\text{Bernoulli}(\Phi(\boldsymbol{x}))$, where $\Phi(\boldsymbol{x}) = \boldsymbol{P}(\boldsymbol{x'} \leq \boldsymbol{x})$ for $\boldsymbol{x'}\sim\boldsymbol{\mathcal{N}}(0,1)$ is the cumulative density function of the standard normal distribution. The GELUs compute the expectation of this value:

$$\text{GELU}(\mathbf{x}) = \mathbf{x} \cdot \Phi(\mathbf{x}) + 0 \cdot (1 - \Phi(\mathbf{x})) = \mathbf{x}\Phi(\mathbf{x}). \tag{1.9}$$

BERT model size variants:

- *base* with 12 layers as seen on Figure 1.5, 12 attention head and hidden size 768, contains more than 110 million parameters total,
- *large* with 24 layers, 16 attention heads and hidden size 1024, contains more than 340 million parameters total.

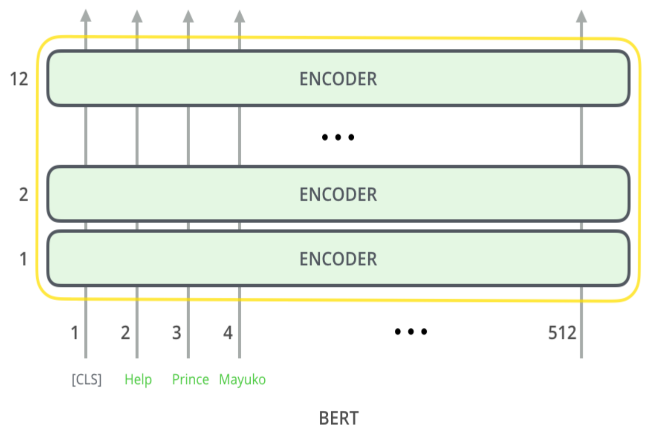


Figure 1.5: BERT encoders input, taken from `http://jalammar.github.io/images/bert-encoders-input.png`.

To pre-train the BERT model, we do not use two independent left-to-right or right-to-left language models like Peters et al. [2018] and Radford et al. [2018]. Instead, to pre-train the BERT model we actually combine two objectives. First is the so-called *masked language model*, and second is the *next sentence prediction*.

Masked Language Model

We want to perform the computation in a bidirectional way, so we cannot just perform language modeling. Language modeling allows us to observe or depend only on words which we have generated. However, we would like the computation to also depend on the right context. This means that you can attend to anywhere in the sentence and you can also attend to yourself. So you can try to predict the current value copied from the input.

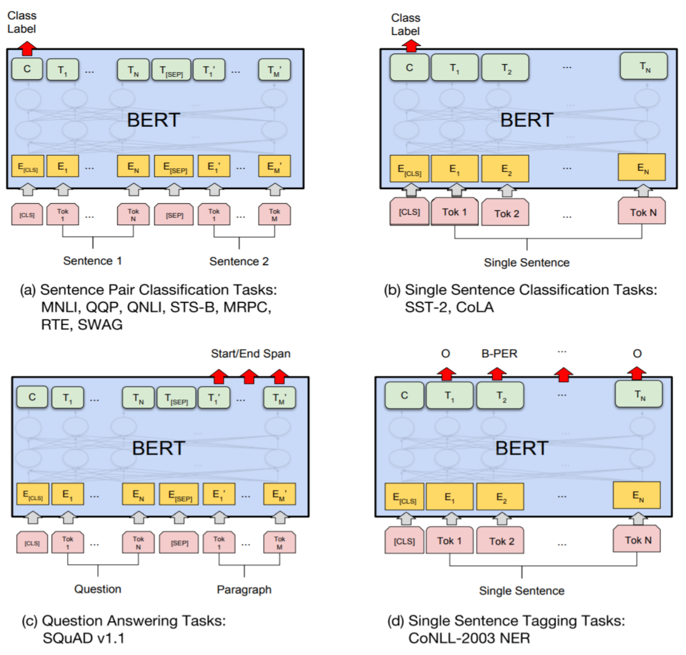
To overcome the problems of a language model approach, the Masked language model was proposed. In this model, for each sentence we randomly select 15% of the input words which are masked, and the model tries to predict them. We have not given the input to the model, and we want to predict them. The problem is that during inference, we do not want to use any mask and therefore we need to do it carefully. The 80% of those 15% of the input words are replaced by a special [MASK] token. However, we need to be prepared for the inference; that's why 10% are replaced by a random word. This means the model will encounter a situation where it needs to compute something which is not masked. The word is replaced by a random word and because the model needs to consider not believing the word on input. The last 10% of those 15% are left intact. These we want to use during prediction when we want to compute representation for every word in input.

Next Sentence Prediction

The goal is to motivate the model to understand the meaning of whole sentences. One way how to do it can be to classify whether two masked sentences mean the same or not. On the Figure 1.4 are these sentences shown as sentence A and sentence B, where A is the paraphrase of B. However, we do not have any data for such a task, just for the plain text, so instead of solving the paraphrase detection, Devlin et al. [2018] propose to solve the next sentence prediction problem instead. The model tries to predict whether the second sentence followed the first one in the raw corpus. In the training data, 50% of the time the second sentence is the actual next sentence, and the gold label is true. The remaining 50% of the time, the second sentence is a random sentence from the corpus and the gold label is false.

1.2.2 Fine-Tuning

Performing the fine-tuning is actually quite simple. Dropout 0.1 is usually fixed, small number of epochs (2-4) is usually sufficient and a good learning rate is usually one of 5e-5, 3e-5, 2e-5. The fine-tuning is straightforward since the pre-trained BERT model, as described previously, can be fine-tuned on a range of tasks showed on the Figure 1.6.



Class Label
BERT
Sentence 1
Sentence 2
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG
Class Label
BERT
Single Sentence
(b) Single Sentence Classification Tasks: SST-2, CoLA
Start/End Span
BERT
Question
Paragraph
(c) Question Answering Tasks: SQuAD v1.1
O
B-PER
O
BERT
Single Sentence
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Figure 1.6: Showing of fine-tuning BERT on individual tasks. (a) Sentence pair classification problem, input sentence pairs and output classification labels. (b) Single sentence classification problem, input sentence, output sentence category. (c) SQuAD question, find out a few words corresponding to a question from a paragraph. (d) Sequence labeling problem, labeling each word in a sentence. Taken from paper Devlin et al. [2018], Figure 4.

1.2.3 Modifications

BERT has many different variants and in this section we mention the most essential ones.

mBERT

The multilingual BERT is pre-trained on 102-104 largest Wikipedias, including the Czech one. Currently are two versions available, the first is the *multilingual cased*, which has WordPieces including case and diacritics, and the second is the *multilingual uncased*, which has all subwords in lower case and without diacritics. Surprising is, that even for languages that have only a small percentage of the training data, it actually works very well. To clarify, only one percent of training data is in the multilingual BERT for Czech language.

The advantage is that without any explicit supervision, mBERT is able to represent more than one hundred input languages in a *shared space*. In the sentence, mBERT understands that the word *dog* and the word *pes* are with the same meaning without seeing any specific dictionary or any specific information in the training data. This representation of multilingual language in the same shared space allows us to perform so-called *cross-lingual transfer*.

The results of the MultiLingual Question Answering (MLQA) dataset for the *reading comprehension* task are published in the paper Lewis et al. [2019], where a question and an answer needs to be located in the paragraph. Data are available in 7 different languages. Training the model in English and then running it on an other different language works comparably to translating the data to English and then back.

RoBERTa

The RoBERTa model, Robustly optimized BERT approach, was firstly introduced in Liu et al. [2019b].

The next sentence prediction [1.2.1] was originally hypothesized to be an important factor during training of the BERT model, as indicated by ablation experiments from Devlin et al. [2018]. Later experiments indicated that removing the next sentence prediction might improve the results. The RoBERTa authors performed the following experiments:

- *Segment-pair:* pair of segments with the most 512 tokens in total.
- *Sentence-pair:* pair of natural sentences, usually significantly shorter than 512 tokens.
- *Full-sentences:* just one segment on input with 512 tokens, can cross document boundary.

- *Doc-sentences:* just one segment on input with 512 tokens, cannot cross document boundary.

The results of individual experiments are summarized in Table 2 in the paper Liu et al. [2019b], where we can see the *full-sentences* approach is pretty good. For that reason, RoBERTa is trained with dynamic masking full-sentences without next sentence prediction with 8 thousands minibatches and byte-level BPE (Byte-Pair Encoding) with 50 thousands subwords.

ALBERT

The ALBERT, A Lite BERT model was proposed in the paper Lan et al. [2019], with small size of the model in mind. The authors consider three main contributions to achieve smaller size:

- **Factorized embedding parametrization:** The authors proposed to represent the subwords using just embeddings of size E and then using a matrix of size $\mathrm{E} \times \mathrm{H}$ to generate the corectly-sized embeddings for the first layer. Originally, in BERT, the subword embeddings had the hidden size dimension H which resulted into a quite large number of parameters.

- **Cross-layer parameter sharing:** The parameters of the soft-attention and the FFN are shared across layers to improve parameter efficiency.

- **Sentence order prediction:** It is an alternative to next sentence prediction. Is considered to have given two consecutive segments and predict which one appeared first in the original document.

We recommend to read the original paper Lan et al. [2019] for more details. You can also find the specific configurations for different versions in Table 1 in the same paper.

2. Datasets

In this chapter, we introduce the datasets used by us. We focus on their origin, basic statistics and properties. The main difference between them is the length of their texts. On the other hand, they are all in Czech language. The first two datasets we use in almost the same form as they are published in the papers Habernal et al. [2013b] and Král and Lenc [2017]. The third dataset is the most interesting, because the form we use has not been published anywhere, and is partly created by us.

2.1 Facebook Dataset

The Facebook dataset was published in 2013 in the paper Habernal et al. [2013b]. As the name suggests, it contains posts from the social network Facebook, especially from the Czech pages, which have the largest fan base. Individual posts (10 000) were randomly selected from nine Facebook pages and, of course, they were anonymized because only text content is used. The pages and their corresponding number of posts are shown in Figure 2.1. The dataset contains 2 587 positive, 5 174 neutral, 1 991 negative and 248 bipolar posts. Bipolar posts will not be used just like the authors of the paper Habernal et al. [2013b], where they mention that bipolar posts are completely omitted from all experiments.

In the Figure 2.1 we can also see that the negative posts mainly relate to pages of telecommunication companies and the positive posts are related only to perfume or ZOO pages.

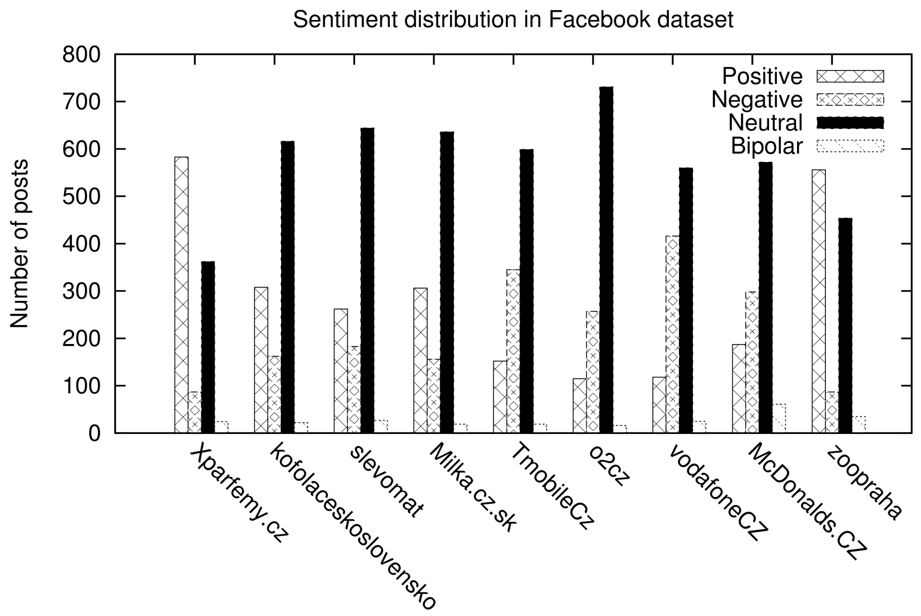


Figure 2.1: Sentiment distribution, from paper Habernal et al. [2013b], Figure 1.

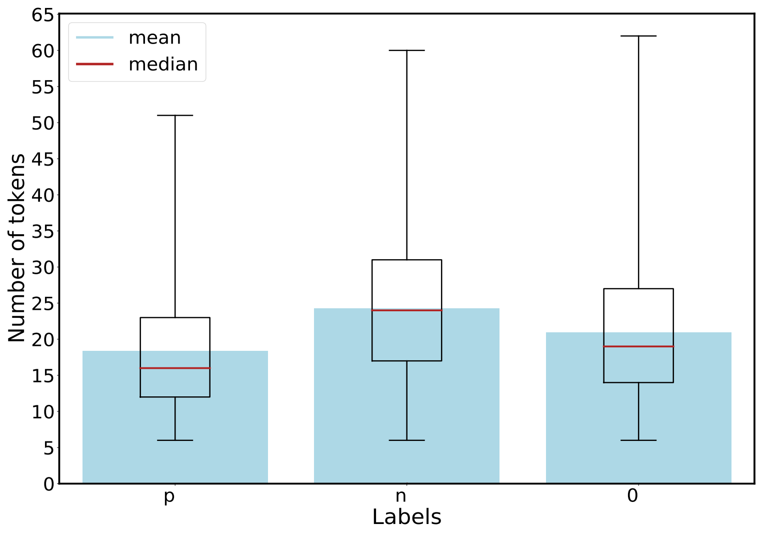


Figure 2.2: Facebook dataset statistics, distribution posts per labels.

Figure 2.2 shows posts per labels statistics, and we can see that the longest posts contain 51 to 62 tokens, and that there are just a few of them. On average are the longest posts negative. The lengths are also symmetrically distributed for these negative posts because the median is almost equal to the average. On the contrary, on average are the shortest posts positive. We can also see that the shortest posts (6 tokens) are present in all labels.

Almost half of the posts are posts with the length of 10 to 18 tokens. Only a few posts are longer than 45 tokens, as shown in Figure 2.3, where the numbers of posts for each length are presented.

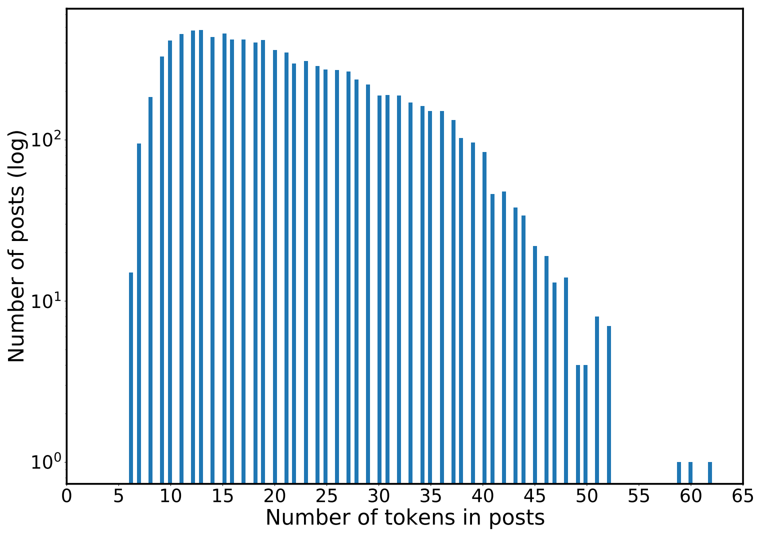


Figure 2.3: Facebook dataset, lengths of posts.

2.2 Czech Text Document Corpus

The Czech Text Document Corpus v 2.0 consists of text documents provided by the Czech News Agency (ČTK) in Czech language. The corpus was published in 2018 in the paper Král and Lenc [2017] and is available at `http://ctdc.kiv.zcu.cz/`. Although its basic version 1.0 is from 2013 Hrala and Král [2013], we mention the properties and statistics from version 2.0 Král and Lenc [2017], which we use. The train set is the same in both versions. It is intended mostly for the automatic classification of documents and is available for research purposes. The main purpose of using this corpus is direct comparison of different approaches to document classification on Czech data. One document is usually labelled with more than a single label, so the corpus can be used for evaluation of multi-label document classification approaches.

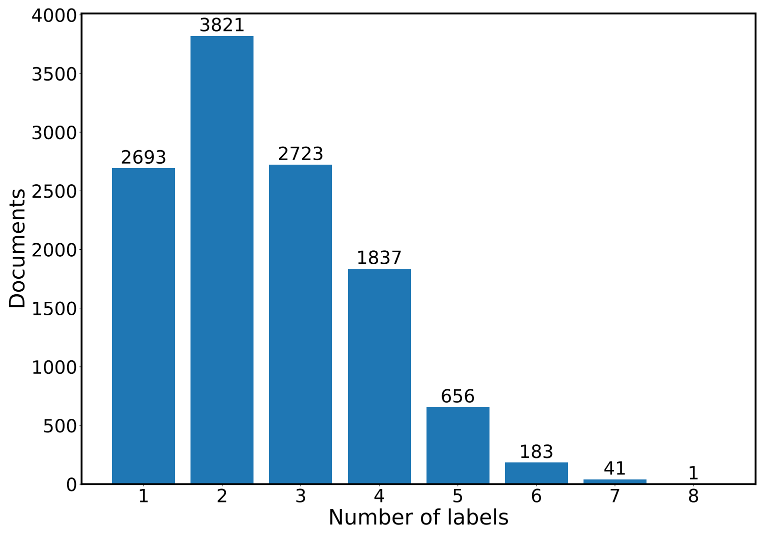


Figure 2.4: Distribution of documents depending on the number of labels, from paper Král and Lenc [2017], Figure 1.

Documents belong to various categories, such as politics (pol), criminality and law (zak), companies (efm). The whole list of categories is shown in Appendix A.1. One or more categories are assigned to each document. Figure 2.4 shows the distribution of documents depending on the number of labels. We can see that most documents have two categories and that up to 8 categories can be assigned to one document. The average number of categories for one document is 2.55. Total number of categories is 60 of which just 37 most frequent are used for classification. On Figure 2.5 we can see the distribution of all 60 categories, where the categories used for classification are marked in orange, and the unused categories are marked in blue. The authors of paper Král and Lenc [2017] state, that the reason for reduction of the number of categories is to keep only the classes with a sufficient number of occurrences for model training.

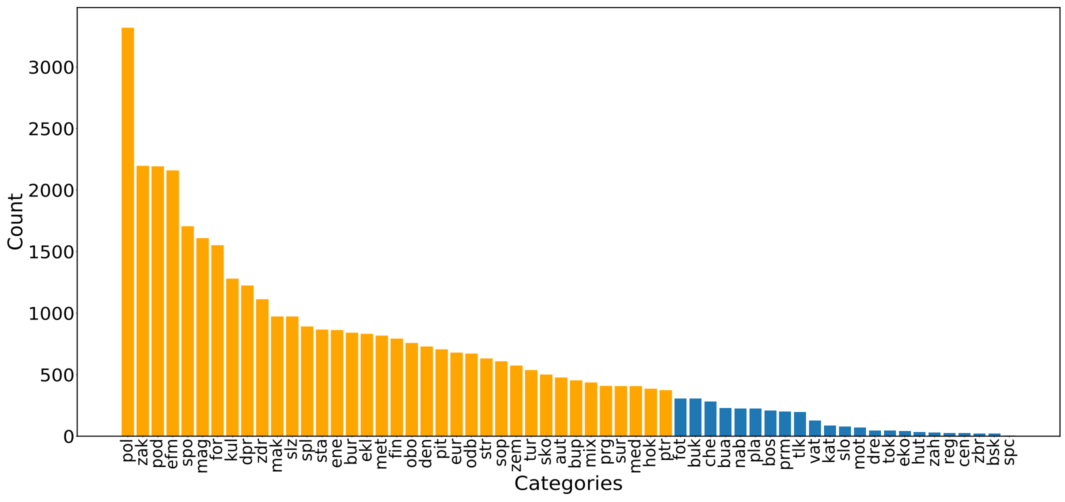


Figure 2.5: Distribution of categories. The most frequent ones are shown in orange, other in blue.

The main part of the corpus (Train) contains 11 955 documents and is intended primarily for training and testing. The development set with additional 2 735 documents is also provided and intended for tuning hyperparameters of the created models. Furthermore, in version 2.0 an annotation on the morphological layer was added. On the Figure 2.6 we can see the length distribution of the individual documents. Blue bars reflect the length of documents depending on the number of words, orange bars reflect it depending on the number of tokens in the document. The orange bars are more important for our work due to the length restriction of BERT. Generally, the documents are quite long, they contain on average 255 words. An interesting observation is that more than 2 000 documents contain at most 50 words. The corpus contains a total of 3 505 965 words, of which 50 899 are unique words, and 82 986 are unique lemmas.

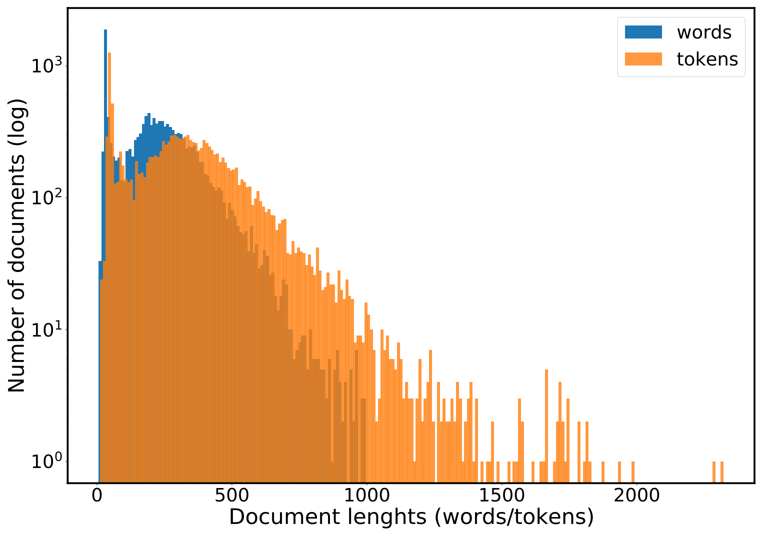


Figure 2.6: Distribution of the document lengths.

2.3 Contracts from the Hlídač Státu

The mentioned datasets, whether Facebook or Czech Text Document Corpus, both fit into one BERT window of 512 tokens, except for a negligible number of documents. The motivation to create a new dataset is, that according to our knowledge, there is no Czech public dataset focused on long texts suitable for low-resource text classification. Utilizing the work of Maroušek [2020], who automatically classified contracts for Hlídač státu[1] in his Master thesis, we create a dataset from the publicly available contracts, annotated with the existing annotation pipeline of Maroušek. The process of obtaining and creating a dataset is shown in Figure 2.7.

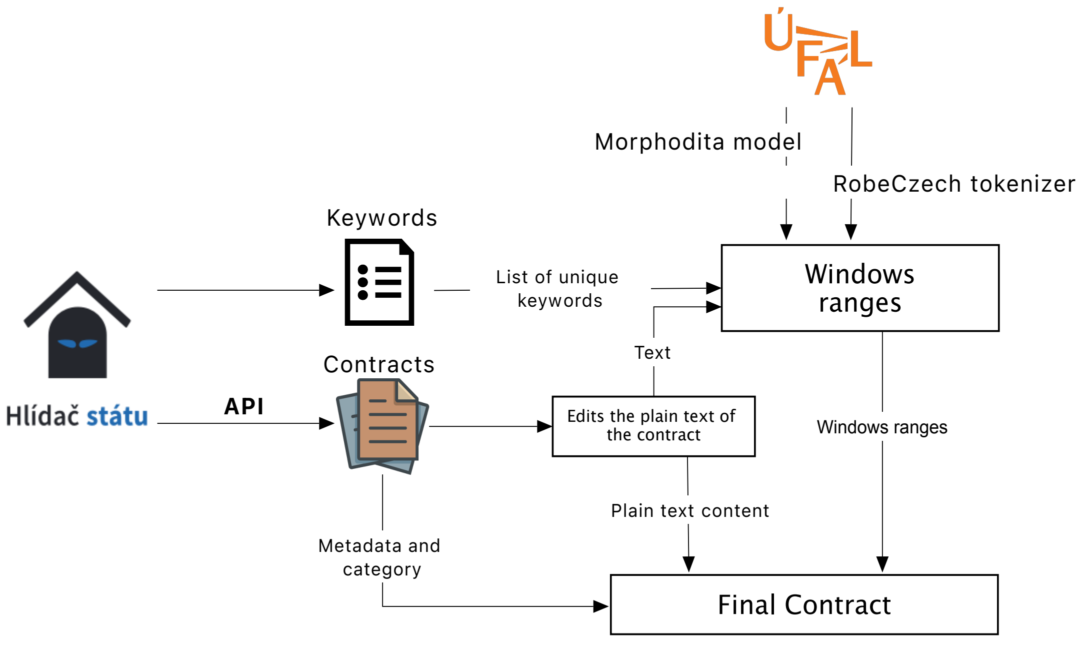


Figure 2.7: Scheme for obtaining final contracts.

2.3.1 Hlídač Státu

The Hlídač Státu project was created in 2016 to unite two original projects, HlidacSmluv.cz and HlidacEET.cz. Its founder and director is Michal Bláha.[2] The objectives of the Hlídač Státu are:[3]

- to make information available about the management of the state and the self-government clearly and comprehensibly,
- to connect and show the mutual relations between state contracts, public procurement, companies and companies with each other, subsidies, sponsors

[1]`https://www.hlidacstatu.cz`
[2]`https://www.michalblaha.cz/o-mne/`
[3]`https://texty.hlidacstatu.cz/o-serveru/`

of political parties and politics,

- to analyze and to allow the public analyze and look at the data of public database,
- to increase the control of public funds by the citizen,
- to identify the waste and abuse of power in the organs.

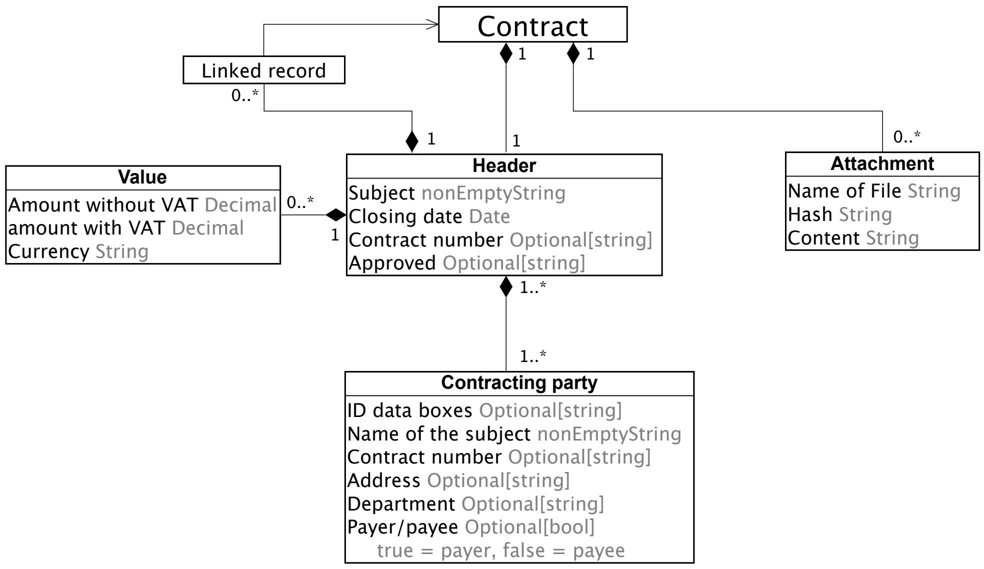
The first step was to download the individual contracts using the REST API HlidacStatu V2.1.1.[4] For each area, we obtained the contract IDs according to their relevance by using the syntax available as a search help.[5] The relevance of the contracts was determined by the Hlídač Státu in its search. We downloaded the first 1 000 contracts for each area, if it was possible. Some areas did not contain so many contracts, so in that case, we downloaded them all. The Hlídač Státu divided contracts into 105 areas at the time we were downloading them. This number has not changed, to our knowledge, while we have been working on this thesis. Therefore, the total number of categories is 105, 22 of them we identify as main categories (Table 2.2) and the rest as their subcategories. The list of categories can be found at the end of this chapter in Table 2.3.

Maroušek [2020] in its classification, which is now used by the Hlídač Státu as a method of keywords, used the specific list of keywords for each area. The lists contain n-grams for n up to 3. The files of these keywords were provided by Hlídač Státu. We use the list of key n-grams, ignoring the categories, which was created by combining these files into one and deleting duplicates. The list of unique keywords contains 49,437 words/n-grams.

2.3.2 Structure of Contracts

Each contract contains a heading that consists of metadata about the contract and attachments. There may be several attachments, which are files with the text of the contract. The heading contains two identifiers. One is unique for each contract record, we call it the identifier of version, and the second is common to those records that relate to a single contract, we call it the identifier of the contract. Other mandatory heading parts are just the subject and the date of conclusion of the contract. Items as the value of the contract or identification of the contracting parties may be empty in eligible cases. The values of contracts are listed with VAT and without VAT, and can be displayed in Czech crowns and also in foreign currency. The contract record structure with UML schema is shown in the Figure 2.8.

[4]`https://www.hlidacstatu.cz/api/v2/swagger/index`
[5]`https://www.hlidacstatu.cz/napoveda`



Contract
1
1
Linked record
0..*
1
1
0..*
Value
Amount without VAT Decimal
amount with VAT Decimal
Currency String
0..*
1
Header
Subject nonEmptyString
Closing date Date
Contract number Optional[string]
Approved Optional[string]
Attachment
Name of File String
Hash String
Content String
1..*
1..*
Contracting party
ID data boxes Optional[string]
Name of the subject nonEmptyString
Contract number Optional[string]
Address Optional[string]
Department Optional[string]
Payer/payee Optional[bool]
true = payer, false = payee

Figure 2.8: Illustrative UML schema of the contract.

In the record, which contains the information about the contracting parties, it is sufficient if only one contracting party is listed for each contract, it is a public institution, to which the contract concerns. All other items, such as entity number, address and data box, may not be shown because they are optional. Using a linked list, each contract may be associated with other contracts, when they are related. The text of the contract has to be attached in each attachment or at least electronic image of the text content of the contract in an open and machine-readable format Maroušek [2020].

2.3.3 Preprocessing the Contract Text

Our approach to preprocessing was very careful, we only removed what was really unnecessary. The text content of contracts was previously described in the subsection 2.3.2, so we know, that it consists of:

- Metadata
- Name and address of the orderer
- Name and address of contractors
- Subject of the contract
- Attachments

For the diversity of future dataset application, we did not decide to combine all parts into one, but rather to keep them as a separate text parts. We only connect the attachments into one text part and we perform just a small edit. In the whole part, we converted all named and numeric characters in the string (e.g., `&gt`, `&#62`, `&#x3e`, ...) with the use of `html.escape()` to the corresponding Unicode characters (Python SW Foundation[6]).

When we were examining individual parts of the attachments, we found that they contain quite a lot of continuous strings formed with a single repeated character. For example, personal data (e.g. a phone number, ...) or similar sensitive data are anonymized using strings such as *"XXXX"* or *"......"* in the text part of the attachment. These characters like *"X"* or *"."* do not generate any word nor any other important unit in any length, so we decided to simplify them into a single character: *"XXXXX"* $\rightarrow$ *"X"*. We performed this operation to strings with at least 3 repeated characters. Therefore, *"AA"* remains *"AA"*, but *"AAA"* we replace with *"A"*. We chose to leave at least one character, because it may be a separation of important text parts in some cases. To further reduce the total length of the attachment and to normalize them, we also merged several continuous spaces into a single one. We remind that any modification was very careful, because our goal was to reduce the text part without losing anything important.

2.3.4 Windows Ranges

Our goal was to obtain sufficiently short sections of the contracts, using the mentioned list of keywords, to determine to which category the contract belongs. By short section we mean a maximum of 512 tokens, in order to fit into one section into the BERT model. Our approach is to use such sections, which contain the most keywords. We assume such segments would contain the most information regarding the meaning. The process of creation of these sections is shown in Figure 2.9 and is described in the next part of the chapter. We first explain the tools used by ÚFAL, and then a method of selecting the best sections of the contracts, in order to concentrate the contained keywords in the middle of the section.

[6]`https://docs.python.org/3/library/html`

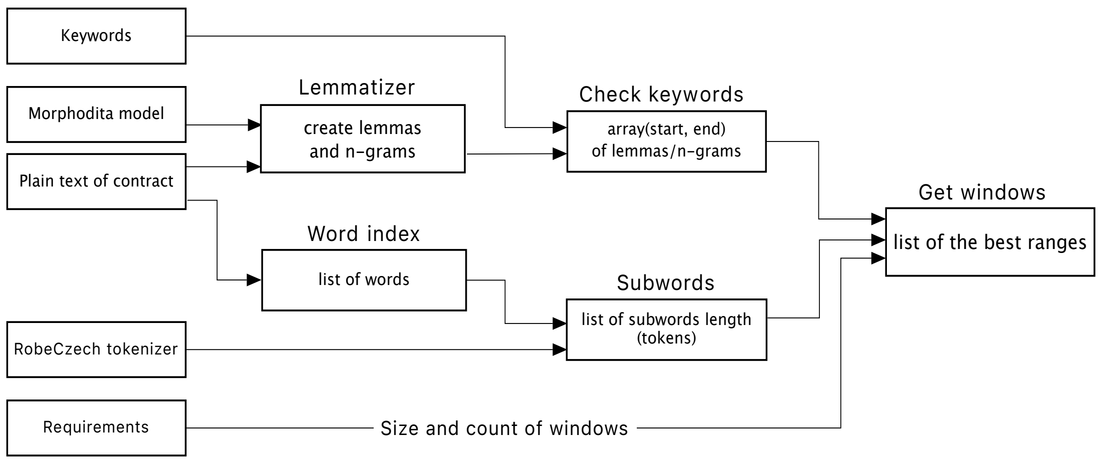


Figure 2.9: Scheme for obtaining windows ranges of the individual contracts.

Lemmatizer

To create lemmas, we use an open-source tool Morphological Dictionary and Tagger (MorphoDiTa) developed by ÚFAL. It is used for morphological analysis of natural language texts, performs morphological analysis with lemmatization, morphological generation, tagging and tokenization. It is distributed as a tool or library with trained linguistic models. MorphoDiTa achieved state-of-the-art results with a throughput around 10-200K words per second Straková et al. [2014] in the Czech language in the year of the article's publication. Nowadays, the LemmaTag system Kondratyuk et al. [2018] using RNNs in lemmatization achieves better results in Czech, but we prefered to use MorphoDiTa, mainly because it is easy to use.

In lemmatization of the contract, we detected word types using MorphoDiTa, and we made the same filtering as Hlídač Státu does, when it searches for the keywords. Specifically, we create lemmas only from nouns, adjectives, verbs and adverbs. Other word types, such as numbers, we replaced with *None* in lemmatization. We chose this filtering, because the n-grams were formed with these word types in the list of keywords.

We did not use any available tool to create the n-grams from lemmatized words, but we implemented it ourselves because it is a straightforward and a simple process. When we say n-gram we mean meta-words consisting of n consecutive words from a given sequence. For a simple example, let's say that in our contract is the phrase *červené auto* (*red car*), using lemmatization we get first lemma *červené* (*red*), the second lemma *auto* (*car*) and by creating n-grams we get also the 2-gram *červené=auto* (*red=car*). When increasing n in the formation of n-grams, we do not lose any information but on the contrary we gain new. In our case, we created a maximum of 3-grams, as the list of keywords did not contain longer n-grams than 3-grams. We decided to create the n-grams only from words that are close to each other in the original text of the contracts, and are not separated by any filtered word type, just like Maroušek [2020]. We did this mainly for consistency and to avoid the formation of unnecessarily big amount of n-grams. For example, from the phrase *Pěkný, Rychlý a Drahý* (*Nice, Fast and Expensive*) we created a 2-gram *Pěkný=Rychlý* (*Nice=Fast*), but not the *Rychlý=Drahý* (*Fast=Expensive*).

Check Keywords

Let's remember that using MorphoDiTa, we return the lemma for each word, and also the range of the lemma. The Check keywords function compares the keywords with lemmas obtained from the text of contracts and if finds a match between them, saving the range (start and end position) of the lemma. This serves as an information, where exactly in the text of the contract an important word or n-gram is located.

Length of Words

The Word Index function is simple but very important function, which cuts the entire text of a contract into individual words. Then we further process these words using the Subwords function by running the RobeCzech tokenizer,[7] a RoBERTa-like tokenizer trained solely on Czech texts, obtaining for each word an array of tokens (without the special tokens like CLS or SEP). We get an array of tokens for each word after tokenizing. The length of the array corresponds to the length of word, which is the number of tokens in the array. For example, for the word *ministerstvo* (*ministry*) we get tokens [29506 568 6633], so the length of word is 3.

Windows

To get the windows we use everything we have already gained, such as the list of lemmas, the list of their lengths, and the exact location of keywords in the text of the contract. All of that is used to obtain the potentially best part of the contract, we hope is the most useful information to help us with the classification. The process begins with the gradual expansion of the sliding window from the first position to the required size specified by the maximum number of tokens. The tokens in the window are represented as a logical mask (1 – keyword, 0 – not keyword) in window. In practice, this is done by checking the array containing the lengths of the words and the list of keyword locations in the text. We choose the window that contains the most keywords. If there is a tie, we decide on the window that has the most central located keywords. This means that the

[7]`https://huggingface.co/ufal/robeczech-base`

margins should be as large as possible to capture as much context as possible. Furthermore, we return the N best windows to provide BERT with multiple segments to process. Once the best window is found, the keywords inside are deleted and searched again until N windows are found. In our case we return 5 windows with a size of 300 tokens for each contract. If the contract is shorter, we return the total number of possible windows. We chose the size of 300 tokens for the following reasons:

- *efficiency* (if we do not fill BERT with the maximum number of tokens (512), we can increase the batch size in our configuration),
- *reserve* (we have the option to add some text to each window).

We complete the whole process of preparing contracts (see the Figure 2.7) by adding the following information to each contract:

- *Label:* category of the contract,
- *PlainTextContent:* text of the contract,
- *WindowsRange:* list of windows ranges.

2.3.5 General Information

The complete dataset contains a total of 97 493 contracts. First, we downloaded 1 000 contracts for each of 105 categories, if it was possible. These categories are all shown in Table 2.3, where we show the specific label for each category like the Hlídač Státu. In total, we obtained 96 469 contracts. Then we randomly took 10% from each category, 9 646 contracts, for the develop set, and the remaining 86 823 contracts are the train set. Therefore, the created dataset has balanced numbers of classes in train and develop set, and thus it does not represent the real number of contracts in individual categories in the Hlídač Státu. Furthermore, they are very noisy for both train and develop set, so when we try to evaluate the system, we do not want to achieve 100% because it will not be the best system. A 100% system would make the same mistakes as the original, which is based on the keyword method.

Furthermore, we asked the director of Hlídač Státu to record the requests from users for changing the contract category. From these records we created the test set. In the period from 29 April 2020 to 26 April 2021, 1 024 requests were recorded, and they form the test set.

Name of set	**Number**
Train	86 823
Development	9 646
Test	1 024
Overall	97 493

Table 2.1: Distribution of contracts according to individual sets.

The 22 main categories, marked in Table 2.3, contain contracts that the Hlídač Státu could not categorize more deeply in the main category. In following graphs, that show the distribution of contracts in the sets, only the main categories are shown, but they are created from all contracts in main categories and also in their subcategories. Showing all 105 categories in graphs would be too much and not clear enough. For a better view, we show the Table 2.2, which contains the names of main categories in English and also in Czech.

Label	Name of category (en)	Name of category (cz)
0	Other	Ostatní
10000	IT	IT
10100	Civil engineering	Stavebnictví
10200	Transport and postal services	Doprava a poštovní služby
10300	Machinery and equipment	Stroje a zařízení
10400	Telco	Telco
10500	Healthcare	Zdravotnictví
10600	Water and food	Voda a potraviny
10700	Safety and protective equipment	Bezpeč. a ochranné vybavení
10800	Natural resources	Přírodní zdroje
10900	Energy	Energie
11000	Agriculture	Zemědělství
11100	Office	Kancelář
11200	Crafts	Řemesla
11300	Social services	Sociální služby
11400	Finance	Finance
11500	Legal and real estate services	Právní a realitní služby
11600	Technical services	Technické služby
11700	Science, research and development	Věda, výzkum a vývoj
11800	Advertising and marketing services	Reklamní a marketing. služby
11900	Other services	Jiné služby
12000	Donations and subsidies	Dary a dotace

Table 2.2: Table of main categories.

The following graphs show a few features of the training set. Figure 2.10 shows the number of contracts in main categories. The more subcategories are in the main category, the more contracts the main category contains. The *Other* category contains only one contract in the training set. In the Figure 2.11 we can clearly see that the contracts are evenly distributed relative to the categories. The average and the median of contracts in the categories remain at one level except for slight fluctuations. This graph further shows the boxplot from which it can be seen, that the contracts are even also in the 4th quartile and that the longest contract is in the category *Transport and postal services*. Figure 2.12 shows the histogram of contracts and we can see that is quite normal with respect to the length of contracts.

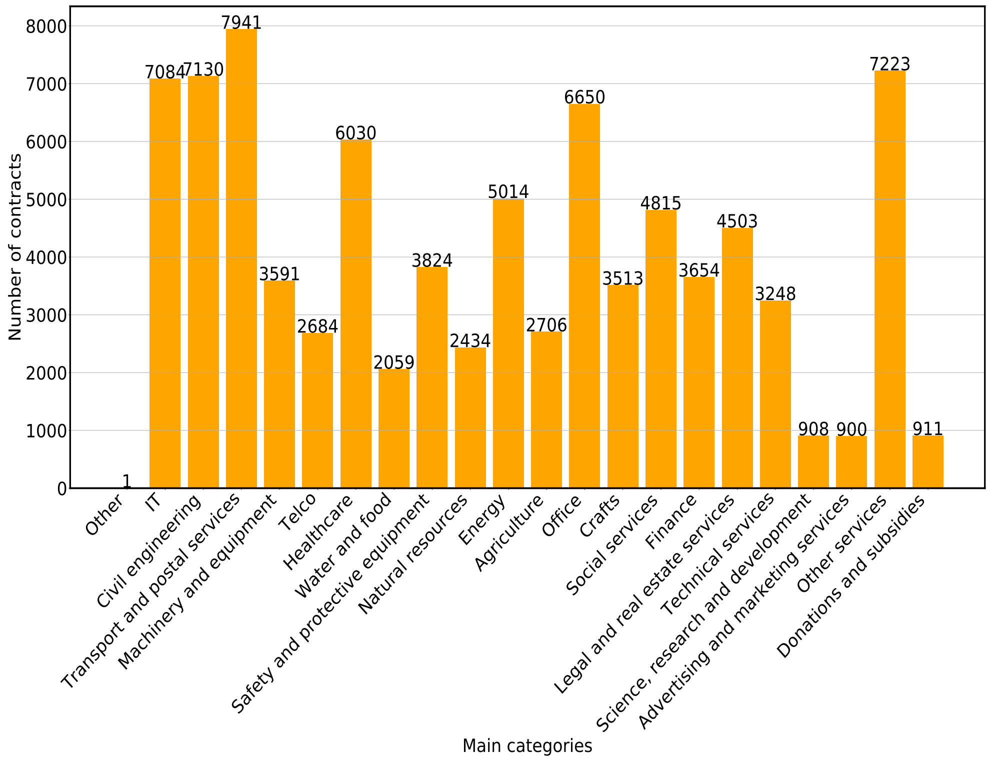


Figure 2.10: Distribution of number of contracts in main categories – Train set.

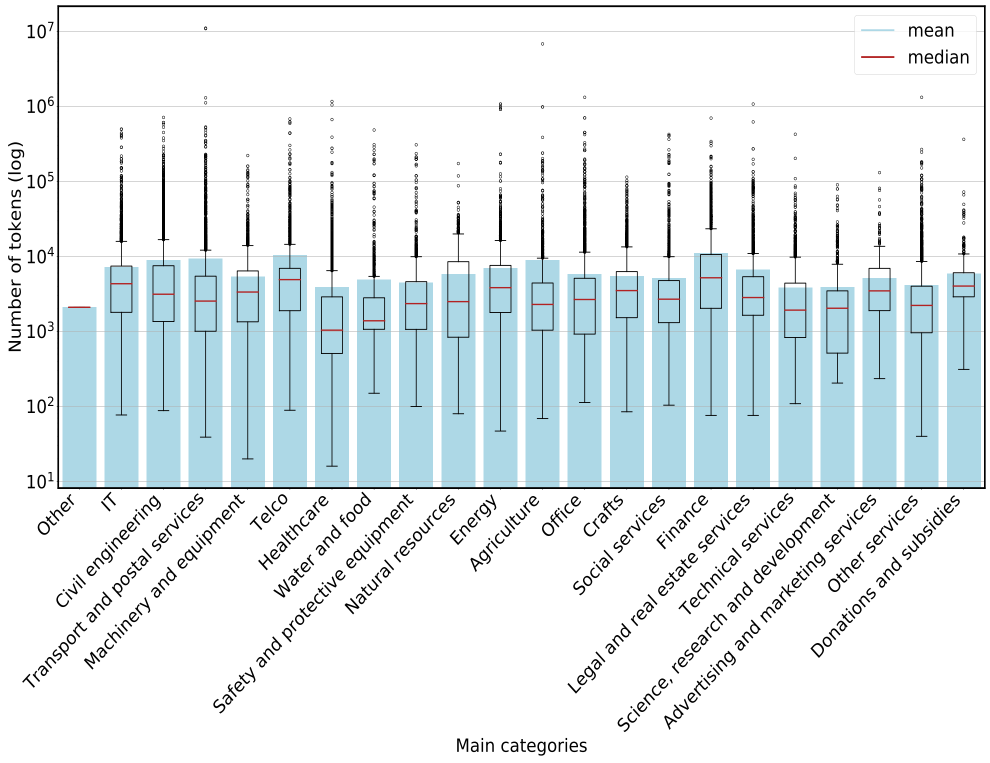


Figure 2.11: Distribution of contracts per categories – Train set.

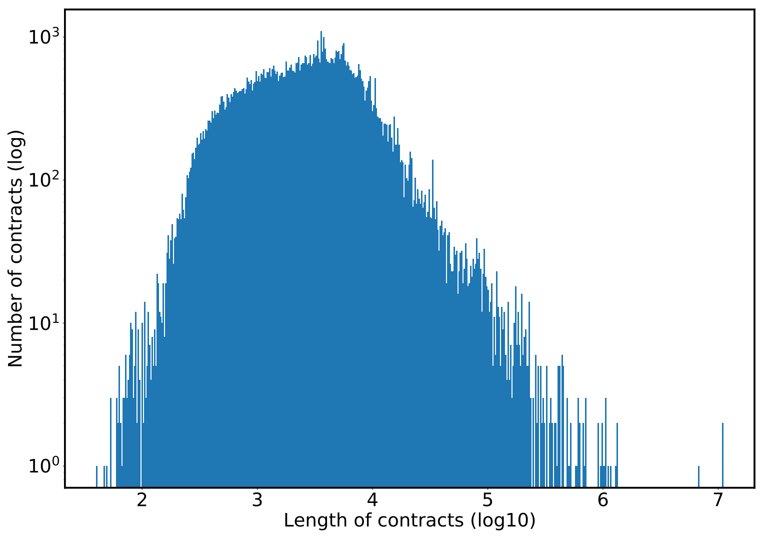


Figure 2.12: Histogram of contracts – Train set.

The following graphs show the information about the test set. In Figure 2.13 we can see that the number of contracts in each category is uneven. These contracts are those that have been misclassified and then corrected by users. It is surprising that only one main category *Natural resources* has no representation. Furthermore, we can see that the most contracts were collected from the *Office* category, and that in categories such as *Water and food, Agriculture, Crafts* and *Science, research and development* there are only a few contracts.

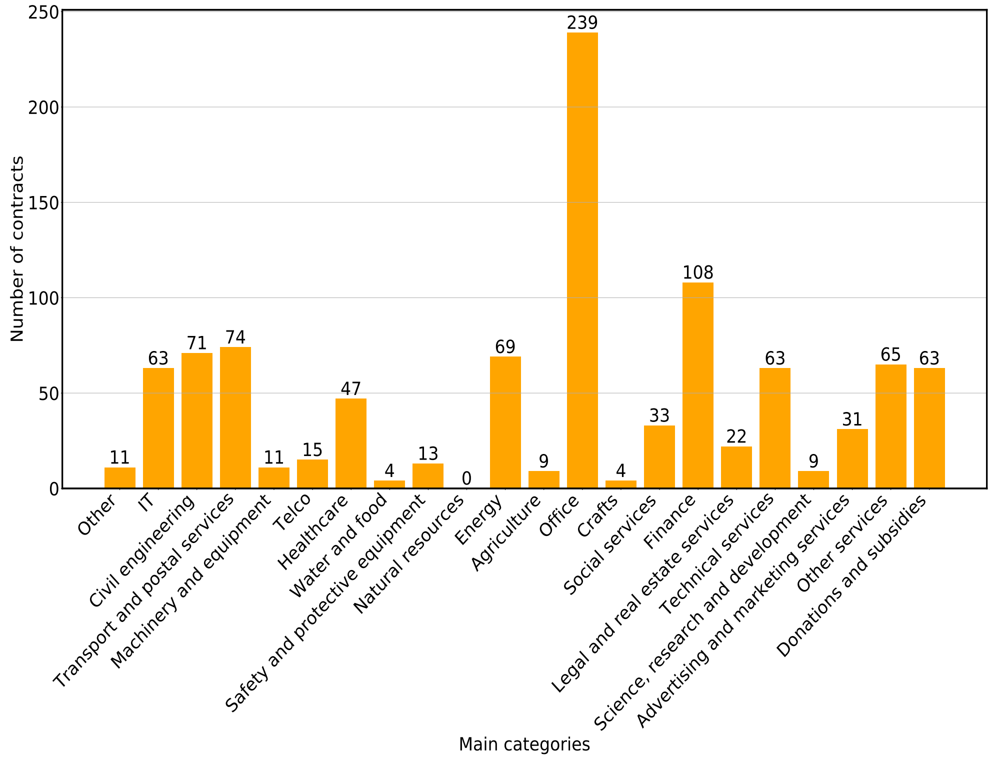


Figure 2.13: Distribution of number of contracts in main categories – Test set.

Figure 2.14 shows that on average the longest contracts are in the *Machinery and equipment* category. It is caused by only two very long contracts of identical length, which are contained in this category. In general, contracts are not as evenly distributed as in the training set. This is probably due to their specific choice and the small number of contracts in each category.

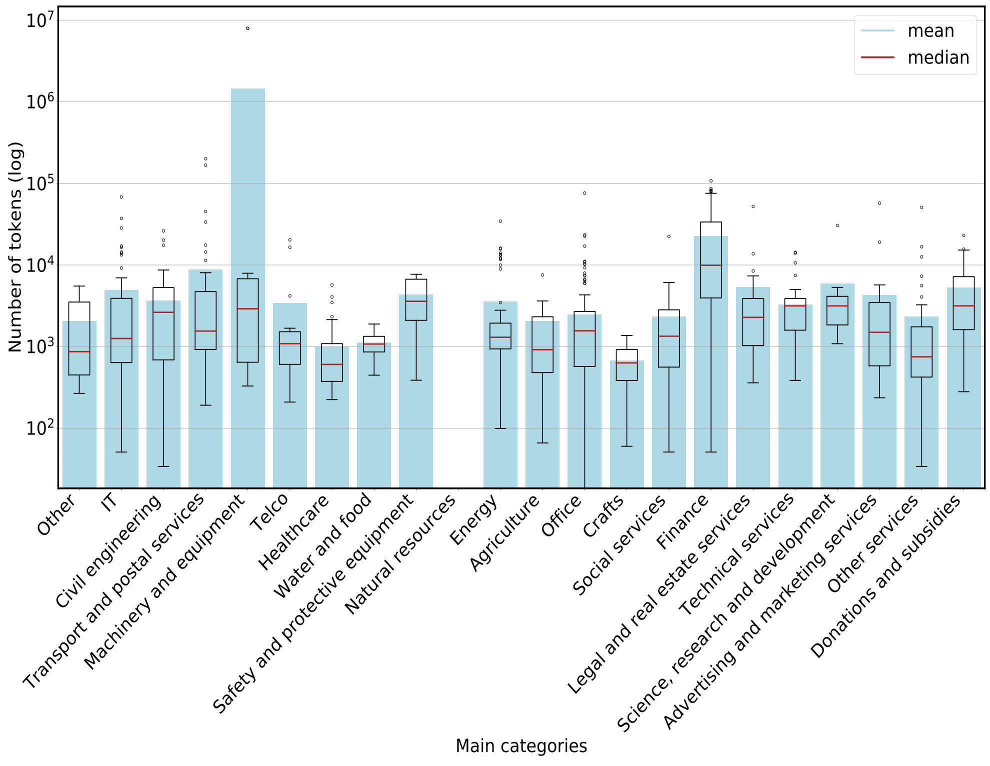


Figure 2.14: Distribution of contracts per categories – Test set.

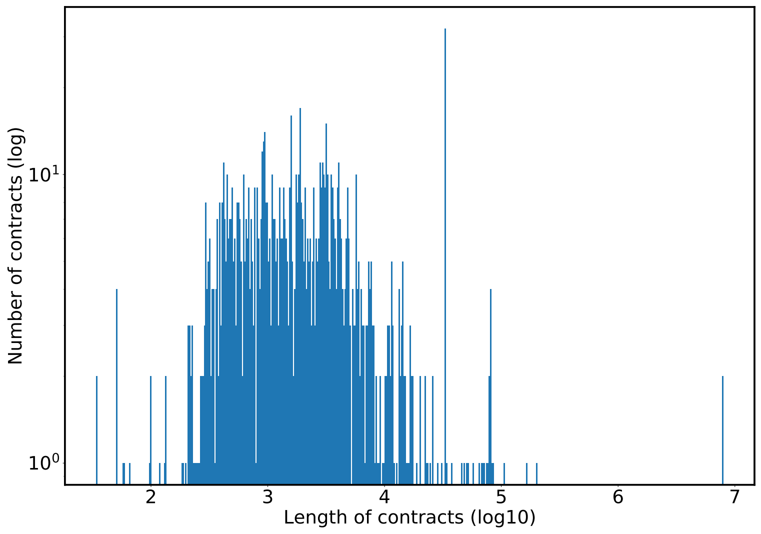


Figure 2.15: Histogram of contracts – Test set.

In the histogram, which is shown in Figure 2.15, we can clearly see long contracts in the mentioned *Machinery and equipment* category. The dominant peak represents 37 contracts with a length of 33 636 tokens. However, despite these few fluctuations, the histogram tends to have a normal distribution.

The reason why we did not present any specific graphs about the develop set is because they would be similar to those about the training set. As a proof, we attach the graph for Cumulative Distribution Function (CDF) for all sets in Figure 2.16. In this graph, we can see from CDF of all three sets (Train, Develop and Test set) that they have a normal distribution and that the develop curve copies the training curve.

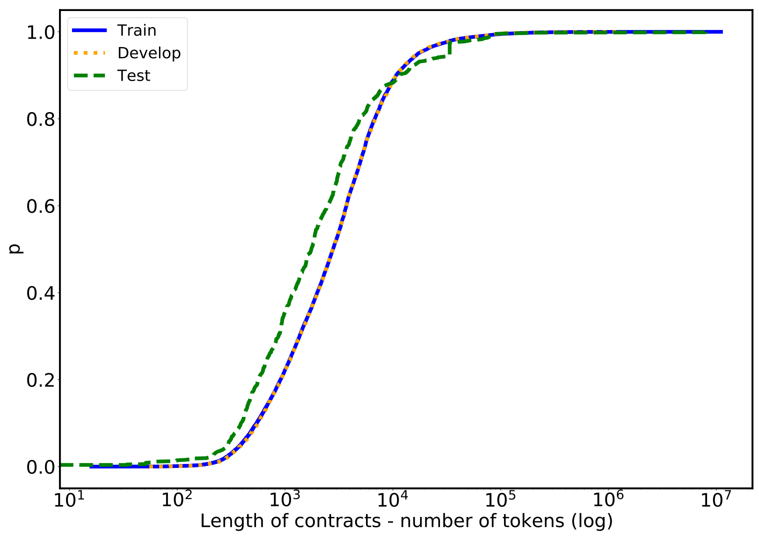


Figure 2.16: Cumulative distribution functions for all sets.

2.3.6 Published Version

Due to the fact that we will evaluate the classification models of the contracts on this dataset, it is advantageous to have a permanently publicly available and citable collection of contracts. Therefore, we decided to save the dataset of contracts created by us in the LINDAT/CLARIN[8] repository, with the permission of Hlídač Státu. LINDAT/CLARIN is a language research center providing a technical background to institutions or researchers to share, create or improve their tools and also provides data which are used in research in the field of linguistics. It provides an open repository and archive that is available to any academic. In addition, storing data in the repository is free, secure and respecting the publication license we have chosen. We publish the dataset under the license Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0).[9] It sets the following conditions:[10]

[8]https://lindat.mff.cuni.cz
[9]https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.txt
[10]https://creativecommons.org/licenses/by-nc-sa/4.0/

- *Attribution* – the need to provide a link to the license and whether there have been any changes. The manner of presentation may be by any reasonable way other than to indicate that the licensor encourages its use.
- *NonCommercial* – the published dataset with that license cannot be used for commercial purposes.
- *ShareAlike* – all works based on this dataset will have to have this license, so any derivatives of the dataset will not be able to be used for commercial purposes.

We publish the dataset of contracts in exactly the final form as we described it in the previous chapters and it is available in this link: `http://hdl.handle.net/11234/1-3731`. The scripts that were used for creating this dataset are available as attachments to this work and their brief description is in Appendix A.2.

Label - Name of category (cz)	Label - Name of category (cz)
0 – Ostatní	**10900 – Energie**
10000 – IT	10901 – Paliva a oleje
10001 – IT Hardware	10902 – Elektricka energie
10002 – IT Software	10903 – Jiná energie
10004 – Opravy, údržba a počítačové sítě	10904 – Veřejné služby pro energie
10005 – IT Vývoj	10905 – Voda
10006 – Konzultace a poradenství	**11000 – Zemědělství**
10008 – Internetové služby, servery, cloud	11001 – Lesnictví a těžba dřeva
10009 – IT Bezpečnost	11003 – Zahradnické služby
10100 – Stavebnictví	**11100 – Kancelář**
10104 – Konstrukční a stavební práce	11101 – Tisk
10106 – Stavební práce pro potrubní, tele. a el.	11102 – Kancelářské potřeby
10107 – Výstavba a práce pro silnice	11103 – Nábytek
10108 – Stavební úpravy pro železnici	11104 – Kancelářské a domácí spotřebiče
10109 – Výstavba vodních děl	11105 – Čisticí výrobky
10111 – Práce při dokončování budov	11106 – Nábor zaměstnanců
10114 – Stavební služby	11107 – Mobily, smart zařízení
10200 – Doprava a poštovní služby	**11200 – Řemesla**
10201 – Osobní vozidla	11201 – Oděvy
10202 – Nákladní nebo speciální vozidla	11202 – Textilie
10203 – Hromadná autobus. a vlak. doprava	11203 – Hudební nástroje
10205 – Vozidla silniční údržby a příslušenství	11204 – Sport a sportoviště
10206 – Sanitní a zdravotnická vozidla	**11300 – Sociální služby**
10208 – Želez. a tram. lokomotivy a vozidla	11301 – Vzdělávání a školení
10210 – Servis a oprava vozidel a příslušenství	11303 – Zdravotní péče
10212 – Poštovní a kurýrní služby	11304 – Sociální péče
10213 – Letecká přeprava	11305 – Rekreační, kulturní akce
10300 – Stroje a zařízení	11306 – Knihovny, archivy, muzea a jiné
10301 – Elektricke stroje	**11400 - Finance**
10302 – Laboratorní přístroje a zařízení	11401 – Pojišťovací služby
10303 – Průmyslové stroje	11402 – Účetní, revizní a peněžní služby
10400 – Telco	11403 – Podnik. a manaž. poradenství
10402 – Sítě a přenos dat	11405 – Bank. služby, operace, poplatky
10403 – Telekomunikační služby	**11500 – Právní a realitní služby**
10500 – Zdravotnictví	11501 – Realitní služby
10501 – Zdravotnické přístroje	11502 – Právní služby
10502 – Leciva	11503 – Nájemní smlouvy
10503 – Kosmetika	11504 – Pronájem pozemků
10504 – Opravy a údržba zdra. přístrojů	**11600 – Technické služby**
10505 – Zdravotnický materiál	11601 – Odpady
10506 – Zdravotnický hygienický materiál	11602 – Čistící a hygienické služby
10600 – Voda a potraviny	11603 – Úklidové služby
10601 – Potraviny	**11700 – Věda, výzkum a vývoj**
10602 – Pitná voda, nápoje, tabák atd.	**11800 – Reklam. a market. služby**
10700 – Bezpeč. a ochranné vybavení	**11900 – Jiné služby**
10701 – Kamerové systémy	11901 – Pohost., ubyt. a maloobch. služ.
10702 – Hasičské vybavení, požární ochrana	11902 – Služby závodních jídelen
10703 – Zbraněy	11903 – Administrat. služby a stravenky
10704 – Ostraha objektů	11904 – Zajišťování služeb pro veřejnost
10800 – Přírodní zdroje	11905 – Průzkum veřejn. mínění a stat.
10801 – Chemické výrobky	11906 – Opravy a údržba
10802 – Písky a jíly	11907 – Překlad. a tlumočnické služby
	12000 – Dary a dotace

Table 2.3: Table of all contract categories

3. Related Work

There are many articles about the datasets mentioned in Section 2, especially Facebook dataset in Section 2.1 and Czech Text Document Corpus dataset in Section 2.2. In this section, we introduce the articles that worked with these datasets first. Specifically, for the sentiment analysis of the Facebook dataset, it is the article Habernal et al. [2013b], and for the Czech Text Document Corpus the article Lenc and Král [2016]. From the more current articles, we discuss the article Sido et al. [2021], which works with both of these datasets. Very important for us is that they used their own variant of the Czech BERT, and therefore is technologically close to our work. Our goal is to show their approach and to describe which architectures they use, and to compare our results with their results. At the end of the chapter, we introduce the concurrent work, which explains the Czech RoBERTa model that we use. The final version of this article will be presented in September of this year at the International Conference on Text, Speech, and Dialogue (TSD), but the pre-print version is already available in Straka et al. [2021].

3.1 Sentiment Classification

The goal of sentiment classification is to assign the correct polarity to a given text. Classes such as positive, negative and neutral are usually used as the polarity indicators. The sentiment classification is also known as the polarity detection and this term is often used in various articles. We introduce features they used in the article Habernal et al. [2013b], and how they approached to the dataset.

Text preprocessing begins with tokenization using the Ark-tweet-nlp tool [Gimpel et al., 2010]. This tool was developed and tested in English, but according to their results, it brings satisfying results in Czech as well. Ark-tweet-nlp tokenizer works well with special characters such as smilies, which are very often used in comments on social networks. Overall, the choice of tokenization is an important step because it significantly affects the sentiment analysis Laboreiro et al. [2010]. The part-of-speech tagging was performed with an internal tool by using the Prague Dependency Treebank (PDT) [Hajic et al., 2006]. The authors removed the stop words using the stopword list from Apache Lucene[1], but they left the diacritics unchanged. They found out that only 8% of the comments were missing. The n-gram features and character n-gram features were used like most similar works, and the minimum n-gram occurrence was set empirically for both to 5 to feature space pruning.

[1]`https://lucene.apache.org/core/`

They used two classifiers for evaluation, Maximum Entropy [Harremoës and Topsøe, 2001] and Support Vector Machines (SVM) [Cortes and Vapnik, 1995]. According to the article, they also tried the Naive Bayes classifier [Vijaykumar and Vikramkumar, 2014], but did not include it in results because they were worse than the two mentioned classifiers.

3.2 Multi-Label Document Classification

Multi-label document classification is a variant of classification, where it is possible to assign more than one label to a document. We focus on the article Lenc and Král [2016], where they work with Czech Text Document Corpus v 1.0 Hrala and Král [2013]. The difference between v 1.0 and v 2.0, which we describe in Section 2.2, is that in v 1.0 the morphological annotation and the development set is missing. These differences have no effect on the classification, and therefore we show their overall approach to the task.

At the time of this work, neural networks in the field of natural language processing (NLP) were very popular, and this popularity still persists. Therefore, the authors used two different neural networks for classification. The first was a standard Multi-layer Perceptron (MLP) [Murtagh, 1991], and the second was a Convolutional Neural Network (CNN) [Fukushima, 2003].

The authors employ a standard Feed-forward Deep Neural Network (FDNN) [Bebis and Georgiopoulos, 1994] with two hidden layers. The input text is represented using Bag of Words (BoW) [Wallach, 2006], so each unique word in the vocabulary is represented with a binary feature indicating its presence in the input. The size of this vector was limited to the N most common words. The documents were just slightly preprocessed, including the conversion of all characters to lowercase, and the replacement of numbers with one common token. The first hidden layer had 1024 nodes and the second one contained 512 nodes. The output layer had size 37 (the number of categories). The values of the nodes in the output layer were thresholded due to the multi-label classification, see the section 3.2 Lenc and Král [2016] for a full description.

Word preprocessing and dictionary for CNN are used similarly to FDNN. The input is a sequence of words in the document, which are represented by indexes in the dictionary. Words that the dictionary does not contain have been deleted from the documents to shorten them, so the longer documents do not lose too many words when a fixed length of a document is set. The authors of the network were inspired by the Kim [2014] network, with the difference that they use uniform size for convolution kernels with dimension 1. The architecture of the used network is shown in Figure 3.1.

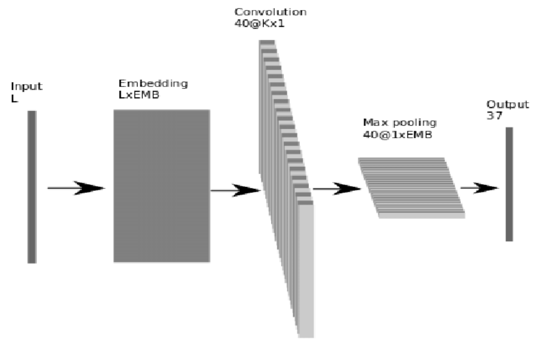


Figure 3.1: Architecture of the convolutional network. Taken from Lenc and Král [2016], Figure 1.

The Figure 3.1 shows the input vector of word indices of length L (fixed length used to represent documents). The second layer is the embedding layer and because of that the document is represented as a matrix. Other layers are a convolutional layer and then a max pooling layer, and output is connected to the output layer. At the end, similar to FDNN, the outputs are thresholded. Further details can be found in the Section 3.3 of Lenc and Král [2016].

3.3 Czert – Czech BERT

At the beginning of this year, the first Czech monolingual models based on the BERT and ALBERT architectures were released at the University of West Bohemia in Plzeň. They presented two models *Czert-A* and *Czert-B*, which were pre-trained on more than 340 000 sentences. This number is really large, as it is 50 times more compared to multilingual models in which the Czech language is also present. Specifically, they used the Czech national corpus [Křen et al., 2016], the Czech Wikipedia corpus and their own crawled Czech news. WordPiece was used as the tokenizer [Wu et al., 2016].

The *Czert-A* model is the smaller one with 12M parameters and is similar to the standard ALBERT$_{\text{BASE}}$. For this reason is also faster. The differences from ALBERT$_{\text{BASE}}$ are that they used the WordPiece tokenizer, they used batch size 2 048 due to cluster limits, and they used their own task to predict the next sentence.

The *Czert-B* is larger, contains 110M parameters, and has exactly the same configuration as BERT$_{\text{BASE}}$ with increased batch size to 2 048. Both Czert models were trained using learning rate 1e-4 with a linear decay using the Adam optimizer [Kingma and Ba, 2014]. More details and a full description can be found in the paper Sido et al. [2021].

3.4 RobeCzech – Czech RoBERTa

RobeCzech is a Czech contextualized model based on Transformer Architecture (Section 1.1) and trained only on Czech data. Actually, we can say that it is the Czech version of RoBERTa [Liu et al., 2019b]. This model is similar to the already mentioned *Czert-B* model, in both cases it is a Czech contextualized model, and they have similar number of parameters – *RobeCzech* has 125M and *Czert-B* has 110M. The difference is that *RobeCzech* is based on RoBERTa, and *Czert-B* is based on BERT. For this reason, we can expect better results, because there is a significant difference in results in English version between them, Liu et al. [2019b]. Publicly available Czech texts were used to train *RobeCzech*, consisting of whole documents, with almost 5G words. Texts sources were:

- corpus of contemporary written Czech – SYN v4 [Křen et al., 2016],
- collection of articles from newspapers and magazines – Czes [Czes, 2011],
- larger documents from the Czech part of the web corpus – W2C [Majliš, 2011],
- texts taken from the Czech Wikipedia.

These texts are then tokenized into subwords using a byte-level BPE tokenizer. The Fairseq library [Ott et al., 2019] with a batch size 8 192 and with a maximum length of each sample 512 tokens was used for training. Adam was used as the optimizer. The learning rate has been used with warmup with the maximum learning rate set to $7 \cdot 10^{-4}$. The whole description of the training process and the results of RobeCzech in comparison with several multilingual and Czech-trained models are available in the article Straka et al. [2021].

4. Training and Results

In this chapter, we introduce and explain the metrics that we use to evaluate the performance of the models. Then we present our model in detail, give reason for the choice of the pre-trained type of BERT model, and present all the features that we have experimentally found to help improve the performance of the model. In the last part, we present our experiments with the results and with the specific values of hyperparameters for the models, which we trained with all datasets mentioned in Chapter 2. In each of the resulting experiments on the mentioned datasets, we also present the previously published results. Without the comparison, we would not be able to decide whether our model is well designed and powerful enough.

4.1 Metrics

Correct evaluation of learned models is one of the most important tasks in pattern recognition. Especially performance metrics are essential for classifying, evaluating the quality of teaching methods and learned models. Therefore, in this section, we present and explain the metrics, which we use for evaluating our results. Classification methods are categorized into single-label or multi-label classification based on the association of the labels to the input samples. All the mentioned metrics have in common that they are based on a threshold value and a qualitative understanding of error. These measurements are used when we want the model to minimize the number of errors. For this reason, these metrics are often used in many direct applications of classifiers [Ferri et al., 2009].

4.1.1 Single-Label Classification

Single-label classification associates the input samples to a unique target label from a set of disjoint labels [Er et al., 2016] and is divided into binary classification and multi-class classification [Tsoumakas and Katakis, 2006]. If the input data is categorized into one of two classes, it is a binary classification. If the input data corresponds to one of more than two target labels, it is the multi-class classification.

Binary Classification

Let's present the binary classification, which is the most basic classification and forms the essential requirement that is necessary for the classification method [Er et al., 2016]. In binary classification, the input data samples are categorized into one of two classes. Therefore, each input data prediction has to fall into one of the categories: correctly/incorrectly classified input A – (True A/False A), correctly/incorrectly classified input B – (True B/False B), Figure 4.1.

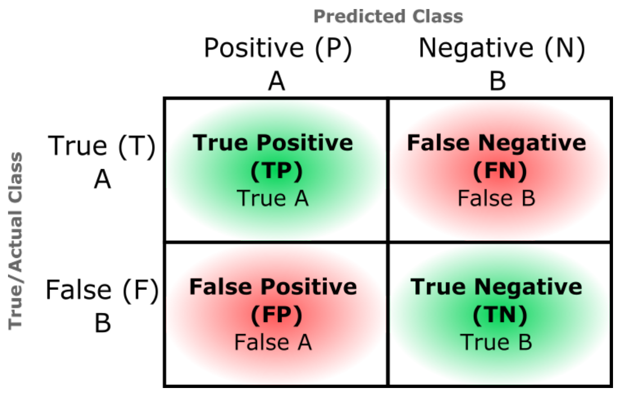


Figure 4.1: An illustrative example confusion matrix for binary classification. There are two true classes true and false. The output of the predicted class is A or B.

Definitions often include notation without using the specific classes, which is also marked in Figure 4.1 and looks like:

- True Positive – (TP),
- True Negative – (TN),
- False Positive – (FP),
- False Negative – (FN),

The result of this classification is called a *confusion matrix*.

- **Accuracy (Acc)** is the safest and the simplest measure to evaluate the classifier. It expresses how correct the predictions of the model are, or, on the contrary, how incorrect is the classification [Ferri et al., 2009].

$$Acc = \frac{|TP| + |TN|}{|TP| + |TN| + |FP| + |FN|} \tag{4.1}$$

- **Precision (Prec)** expresses the ratio of correctly predicted positives in all predicted positives.

$$Prec = \frac{|TP|}{|TP| + |FP|} \tag{4.2}$$

- **Recall (Rec)** is an additional metric of accuracy and expresses the ratio of all positives that were correctly predicted.

$$Rec = \frac{|TP|}{|TP| + |FN|} \tag{4.3}$$

- **F1 score (F1)**, also known as the F-measure, expresses the harmonic average of precision and recall in the range from 0 to 1.

$$F1 = \frac{2 \cdot Prec \cdot Rec}{Prec + Rec} \tag{4.4}$$

Furthermore, we have to emphasize that the correct choice of metric is very important, and that the metric will change when we exchange the True/False classes. Overall, selecting a metric for a task determines what is actually optimized. For example, the F1 metric is often used in competitive tasks because it is a compromise between the best precision and the best recall.

Multi-Class Classification

Multi-class classification means that the input data corresponds to one of more than two target labels. It can be classified into three groups:

- extended methods from binary classification,
- decomposition to binary classification methods,
- hierarchical classification methods [Er et al., 2016].

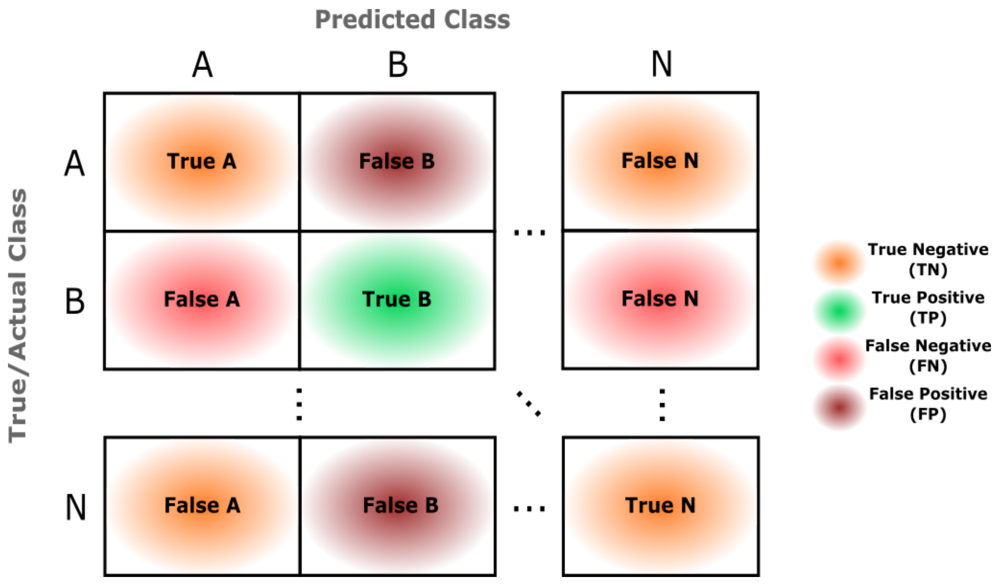


Figure 4.2: Confusion matrix for N-class classification. We can obtain the four different classification results: True Negative – TN (orange), True Positive – TP (green), False Negative – FN (red) and False Positive – FP (dark red).

In binary classification, it matters which class is selected as positive, otherwise a problem with asymmetry may occur. To prevent this problem, multi-class classification can be used. In the section on binary classification, we have shown the confusion matrix for two classes, and now we extend this approach to the classification of N classes. The formulas of individual notations, TP, FP, TN, FN, remain the same as in binary classification. Figure 4.2 shows an example of the confusion matrix for the classification of N classes, which shows the mentioned notations (TP, FP, TN, FN) for classification of class B at position [B, B]:

- TP – the value of the green field (the [B, B] position),
- FP – the sum of the dark red fields (the column B without the TP field),
- FN – the sum of the red fields (the row B without the TP field),
- TN – the sum of the orange fields (everything else without the column B and the row B).

Macro-F1 (macro-averaged metric, macro-recall) is an often used but very simple way for obtaining one value for the metric of multiple classes. It is calculated as an arithmetic mean of the individual classes. It treats all classes equally, ie. in the case of underrepresented class, the metric will affect the model to pay more attention to that class.

Micro-F1 (micro-averaged metric, micro-recall) is also an often used but not as much as the F1 macro method. It is also the way for obtaining one value for the metric of multiple classes and is calculated as a weighted average of the individual classes. It is recommended especially when we want to optimize the overall accuracy of the model.

4.1.2 Multi-Label Classification

In contrary to the single-label classification, in the multi-label classification each input sample can be assigned multiple labels from a set of target labels. The number of target labels corresponding to each input do not have to be fixed, and may change dynamically [Er et al., 2016]. We consider the multi-label classification to be a generalization of multi-class classification, where we actually want to categorize one instance into exactly one class of more than two classes. The multi-label classification has no limit on how many classes an instance can be assigned to.

4.1.3 k-Fold Cross-Validation

The cross-validation method solves the problem of overlapping test sets in testing by validation with repetition [Berrar, 2019]. In k fold cross-validation, the dataset is divided into k disjunct parts of approximately the same size. Only one of these parts is used as a validation set in each iteration and the other $k - 1$ parts are used for training [James et al., 2013]. This iteration is repeated, until each part is used as a validation set. The total performance of this method is obtained as the average of k performance measurements on the individual validation sets [Berrar, 2019]. Only one parameter, k, need to be selected, it is typically 5 or 10. The advantage of the method is a relatively accurate estimate of the classification performance because each part of the dataset appears in both the training and the test set. On the other hand, the disadvantage is the increase of required computation resources. The k-fold cross-validation can be described with the following pseudocode:

```
1. The dataset is shuffled randomly.
2. The dataset is divided into $k$ approximately equal parts.
3. For each obtained part:
    • The remaining $k - 1$ parts are taken as a training set.
    • The model is trained on the training part and evaluated on the validation part.
    • We remember the evaluated score and we no longer need the model.
4. The final evaluation is made by averaging the obtained score evaluations.
```

4.2 Our Model

When choosing a pre-trained model, we first considered the multilingual XML-RoBERTa large model [Liu et al., 2019b]. In the initial experimental demo tasks we achieved very good results. The problem was with its size because we did not have enough resources for using it in the resulting model. We ran the demo tasks on publicly available graphics processing units (GPUs) from Google Colaboratory – Colab[1], where we did not have a stable access to the required number of GPUs needed for the XML-RoBERTa large model. Our resources were limited with the resources from the Artificial Intelligence Cluster (AIC) ÚFAL, where we had access to eight GeForce GTX 1080 graphics cards, each with 8GB of RAM. For this reason, we decided to use a smaller, but comparably powerful, Czech RoBERTa – RobeCzech model [Straka et al., 2021] (more in the Section 3.4) with appropriately performed fine-tuning. RobeCzech clearly outperforms all known similar-sized models, and improves the state-of-the-art results on most NLP tasks we evaluated. Only the already mentioned XML-RoBERTa large overcomes RobeCzech on some tasks, but it is four times as large.

We do not use the officially released checkpoint of RobeCzech from `https://huggingface.co/ufal/robeczech-base`, because it was not available at the

[1]`https://colab.research.google.com/notebooks/intro.ipynb`

time we performed the experiments. Instead, we use a development version of the model, which achieves very similar results according to the internal evaluation of the model's authors.[2]

Our model starts with a text classification architecture using the RobeCzech model, which is based on the BERT encoder (Section 1.2). Then follows the activation function which is activated with a *softmax* or a *sigmoid* function. The choice depends on the number of classification classes, which can be assigned. The *softmax* we use for a single-label classification and the *sigmoid* for multi-label classification. This layer processes the obtained embedding of the given text from the CLS token embedding from the last layer. In our model, we use Adam optimizer algorithm with default parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$) [Kingma and Ba, 2014] with a batch size in the range from 8 to 64. The choice of a specific batch size is determined by the limit of capacity of used GPUs. We use two GPUs in parallel, using a distributed mirrored strategy from TensorFlow [Abadi et al., 2015], due to the possibility of using a larger batch size for training.

We train only the classifier with the default learning rate of 1e-3 to optimize the parameters in the first epoch or first two epochs. We can imagine this process as freezed BERT encoder in the first epoch or first two epochs. The training itself then begins from the second or third epoch, where the whole model is updated. We use the triangle schedule (Section 4.2.1), in which a certain number of epochs form a warm-up and the remaining epochs form a linear decay. This fully linear optimization in both directions was inspired by Liu et al. [2019b], where it was used in training models using RoBERTa for tasks such as the SQuAD or the General Language Understand Evaluation (GLUE). The specific number of epochs for warm-up, the type of metric in the evaluation and other hyperparameters will be present later in the evaluation of the results of each task.

We use early stopping as a regularization technique to prevent pre-training of the model. The early stopping technique consists of monitoring the performance of the model on the development set after each epoch, and conditionally terminating the training by performing validation, Géron [2019].

All used scripts for training and evaluation are included in this work as attachments and their brief description is in Appendix A.3.

4.2.1 Triangle Schedule

The triangle schedule is divided into two parts which are warm-up and linear decay.

Warm-up is the phase at the beginning of training where we start with a learning rate of 0 and increasing it linearly over several iterations or epochs until

[2]Personal communication with the authors.

we reach a predetermined size of the learning rate (the peak learning rate). In other words, it is a linear increase in learning rate depending on the number of epochs and updates, [Liu et al., 2019a, Peltarion, 2021].

Linear decay is the second part of the triangle schedule, which linearly reduces the learning rate by the same decrement in each update, Peltarion [2021].

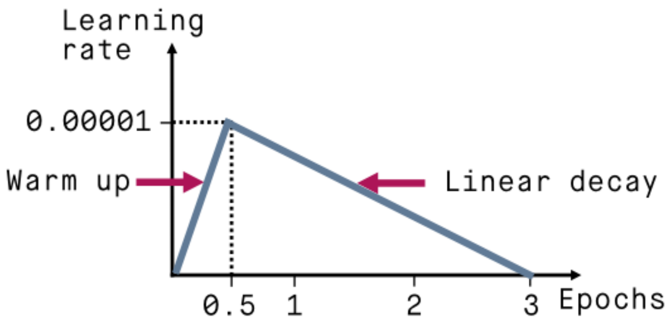


Figure 4.3: Illustration of a triangle decay with the peak learning rate set to 1e-5 at 0.5. Figure is taken from `https://peltarion.com/static/triangle_decay_learning_rate_schedule_a.png`.

The ratio between these phases is variable, but usually the linear decay phase forms a much larger part of the whole training. The aim of the triangle schedule is to enable the model to continuously adapt to the given better parameters, before performing larger number of parameter updates.

4.3 Results

4.3.1 Sentiment Analysis – Facebook Dataset

We evaluate the sentiment analysis on the Facebook dataset [Habernal et al., 2013a] which we described in more detail in Section 2.1. The published dataset contains only one large set of data and does not contain a test set. Therefore, we divided the whole dataset into the train, development, and test sets preserving the original dataset class distribution, similarly to Sido et al. [2021].

We described the basis of our model in the previous Section, but we fine-tuned it specifically for this classification. We tried many configurations, we experimented with the batch size between 16, 32 and 64. We got a slightly better result and the training was faster with batch size 64. When selecting the learning rate peak, we tested 5e-6, 1e-5, 2e-5, 3e-5 and 5e-5. We also looked for the best ratio between warm-up and linear decay and we opted for 1:9. We set the number of output neurons according to the number of classes, in this case it is 3. We use the *softmax* activation for the output layer, due to the fact that it is classified using only a single class. We used *cross-entropy* as a loss function. The final setting of the hyperparameters can be found in Table 4.1. We evaluate the performance using both *accuracy* metric and *macro F1 score*, in order to be able to relevantly compare with other published results.

Batch size	64
Learning rate peak	1e-5
Dropout	0.1
Epochs	50
Frozen epochs	1
Warmp-up	10%
Number of labels	3

Table 4.1: Table of hyperparameters for fine-tuning.

We started by analyzing the dependence of the accuracy of the system on the size of the train set on this dataset. This means that we randomly took a certain amount of data from the original train set, and trained the model on that reduced train set. The development and test set we left intact. We reduce the amount of train set by 10% in the first run. The results are shown in Table 4.2. We can see that even though we trained the model with only 10% of the data from the original training set, the resulting performance (Acc = 78.9, F1 macro = 77.08) is still competitive, given that the model performance is still better than most published results, see Table 4.4. For this reason, we decided to continue reducing the amount of training data by 1%, until we trained the model with only 1% of the training data from the original train set. We can see in Figure 4.4 that the breaking point for the Facebook dataset is at 10% of the training set, and then the performance starts to decrease rapidly.

	1%	**2%**	**3%**	**4%**	**5%**	**6%**	**7%**	**8%**	**9%**
Acc	53.9	61.4	67.8	71.0	71.8	73.4	74.1	76.8	76.9
$\mathrm{F1}_{\mathrm{macro}}$	48.36	55.81	65.96	69.98	71.09	72.36	72.42	74.10	75.22
	10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
Acc	78.9	78.4	79.5	80.1	78.8	80.9	82.3	81.8	81.8
$\mathrm{F1}_{\mathrm{macro}}$	77.08	76.70	77.41	77.81	76.90	78.72	80.49	79.25	79.97
	100%								
Acc	82.6								
$\mathrm{F1}_{\mathrm{macro}}$	81.13								

Table 4.2: Results with respect to the amount of training data.

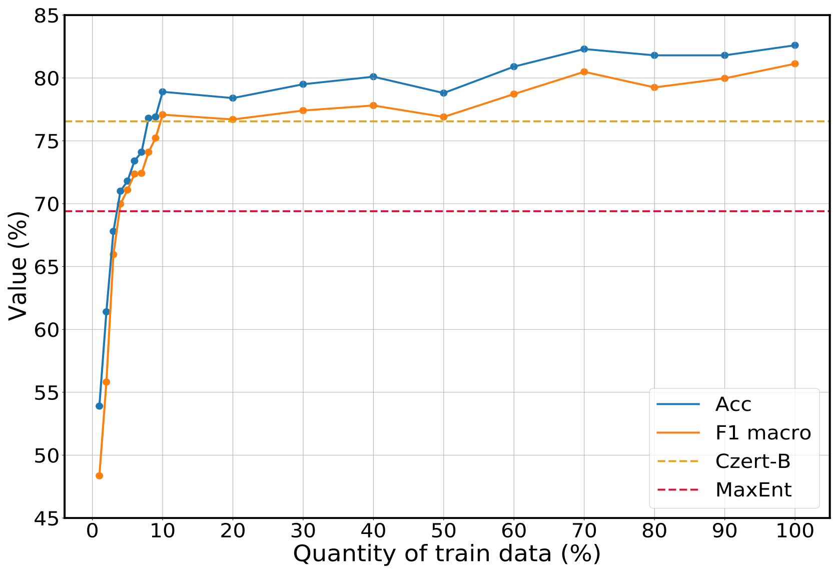


Figure 4.4: Graph of results with respect to the amount of training data.

Our next experiment on this dataset was to study the dependence of the accuracy of the system on the quality of the train set. By quality we mean the correctness of annotation. Reduction of data quality was achieved by adding a random noise of a certain amount to the train and the development set. In this case we also left the test set untouched. By adding noise we mean changing a label to any other incorrect label. In the case of a positive label, there was a random change to neutral or negative. We trained models using train sets where we were reduced the quality of data gradually by 5%. In the most extreme case, we added a noise up to 80% of the train and development set. The performance of the model keeps quite high until the noise of 20%, with accuracy being reduced only by 1.1%-1.7% compared to the initial value. The effect can be nicely observed in Figure 4.5. Another drop occurs between 25% and 45% of noisy level. Surprisingly, the performance of the model is relatively even in this range, but with a more significant decrease compared to the original performance. The biggest drop in performance occurs after the 60% of noisy data, where the performance starts to be very unstable and considerably worse. However, we can state that the model we built on the Facebook dataset with the 45% noisy data achieves results that still exceed some published results obtained with methods, such as SVM or Maximum Entropy, see Table 4.4. Table 4.3 shows all measured values at a certain amount of data noise.

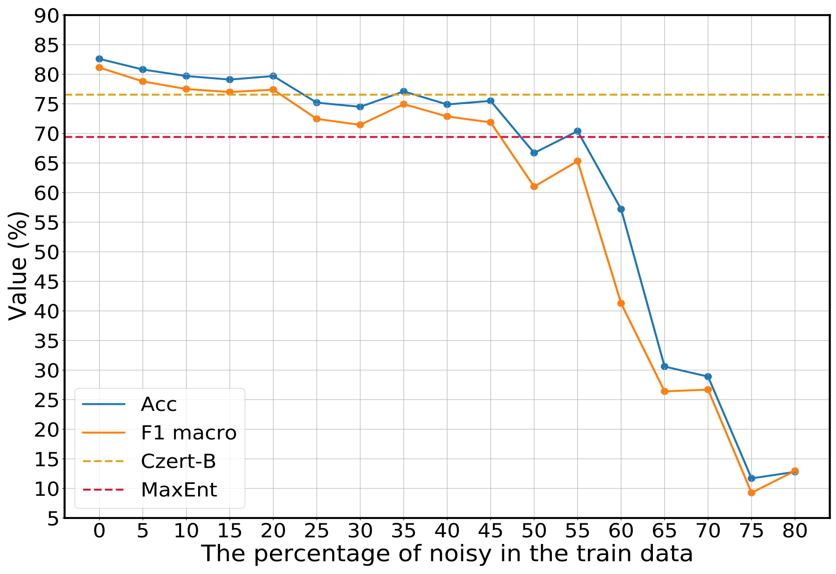


Figure 4.5: Graph of results due to the amount of noisy data.

	5%	**10%**	**15%**	**20%**	**25%**	**30%**	**35%**	**40%**
Acc	80.8	79.7	79.1	79.7	75.2	74.5	77.1	74.9
$F1_{macro}$	78.79	77.50	76.99	77.40	72.46	71.45	74.95	72.87
	45%	**50%**	**55%**	**60%**	**65%**	**70%**	**75%**	**80%**
Acc	75.5	66.7	70.4	57.2	30.6	28.9	11.7	12.8
$F1_{macro}$	71.87	61.01	65.33	41.28	26.40	26.70	9.26	12.97

Table 4.3: Results due to the amount of noisy data.

In Table 4.4 we show our achieved overall result in comparison with models that have comparable size or that are in the original published article of this dataset. We note that the results obtained using the Czert models are the averages of six experiments and in the case of RobeCzech it is a 10-fold cross-validation.

Method	$\mathbf{F1_{macro}}$
SVM [Habernal et al., 2013b]	68.0
MaxEnt [Habernal et al., 2013b]	69.4
Czert-A [Sido et al., 2021]	72.47
Czert-B [Sido et al., 2021]	76.55
RobeCzech [Straka et al., 2021]	80.13
Our model	**81.13**

Table 4.4: Results for sentiment analysis in Facebook datasets.

4.3.2 Multi-Label Classification – Czech Text Document Corpus

We took the Czech Text Document Corpus dataset described in Section 2.2 to verify our model for multi-label classification. We start again from our already mentioned model (Section 4.2) with the appropriate fine-tuning. We use *sigmoid* in this case as the activation function in the output layer and we employ the *binary cross-entropy* function as a loss function. The BERT is limited to a maximum sequence length of 512 tokens, so we had to reduce some documents because they were longer. During the experiment we found that if we use the whole 512 tokens, we cannot set the batch size to a correspondingly large value because of the capacity of the GPUs, and therefore the overall result was unnecessarily worse. Specifically, we were able to use a maximum batch size 8 (with 2 GPUs in parallel as mentioned earlier). For this reason, we tried using less tokens from the text to be able to increase the batch size. We decided for the first 300 tokens from each document and for the maximum batch size, which we could set, which was 16. Compared to the sentiment analysis task, we increased the learning rate a bit, as well as the ”freezing” of BERT was to increased to 2 epochs, allowing the classifier to train longer, because there are more categories now and we do multi-label classification. The final hyperparameters are shown in Table 4.5.

Batch size	16
Learning rate peak	2e-5
Dropout	0.1
Epochs	50
Frozen epochs	2
Warmup-up	10%
Number of labels	37

Table 4.5: Table of hyperparameters for fine-tuning.

We use 5-fold cross-validation for this experiment and the micro F1 score metric to evaluate the performance of the model. The results of the individual folds are shown in Table 4.6. The average of them (89.46) is our overall result for this experiment, and this value we compare with other published results on this dataset in Table 4.7.

	fold 1	**fold 2**	**fold 3**	**fold 4**	**fold 5**
F1_{micro}	89.73	89.58	89.79	89.26	88.93

Table 4.6: Results of individual folds.

Method	**$F1_{micro}$**
MLP [Lenc and Král, 2016]	83.9
CNN [Lenc and Král, 2016]	84.7
Czert-A [Sido et al., 2021]	82.27
Czert-B [Sido et al., 2021]	85.06
Our model	**89.46**

Table 4.7: Results for Multi-label classification in Czech Text Document Corpus.

We remind that both Czert models used the maximum capacity of BERT sequence length (512 tokens). Thus, we have decided well that we prefer the batch size over the use of the maximum amount of text, as we exceed the best state of the art by more than 4%.

We again studied the dependence of the model performance on the size of the training set. For performance reasons, we no longer use 5-fold cross-validation in this case. We start from fold 1 and thus the initial train set as well as other development and test sets are from fold 1. We reduced the size of the training set by 10% in each measurement. The results from each measurement are shown in Table 4.8. We can see from them that a larger drop in model performance for this dataset occurs earlier than with the Facebook dataset. We have a decrease of about 4% at 30% amount of training data. The Facebook dataset had similar decrease at 10% of the amount of training data. Furthermore, it can be seen that this first fold is competitive even with 30% of the training data, in comparison with the overall results shown in Table 4.7. We visually show these results in Figure 4.6, where we can nicely see that if we discard 20% of the training data and train only with the remaining 80%, we still get a result above 89% and the decrease from the original value is minimal. In general, we get the result of over 80% even with 20% of the training data, and then the performance of the model begin to decrease rapidly.

	10%	**20%**	**30%**	**40%**	**50%**
$F1_{micro}$	69.36	81.45	85.40	86.45	87.50
	60%	**70%**	**80%**	**90%**	**100%**
$F1_{micro}$	87.24	88.41	89.15	89.33	89.73

Table 4.8: Results with respect to the amount of training data.

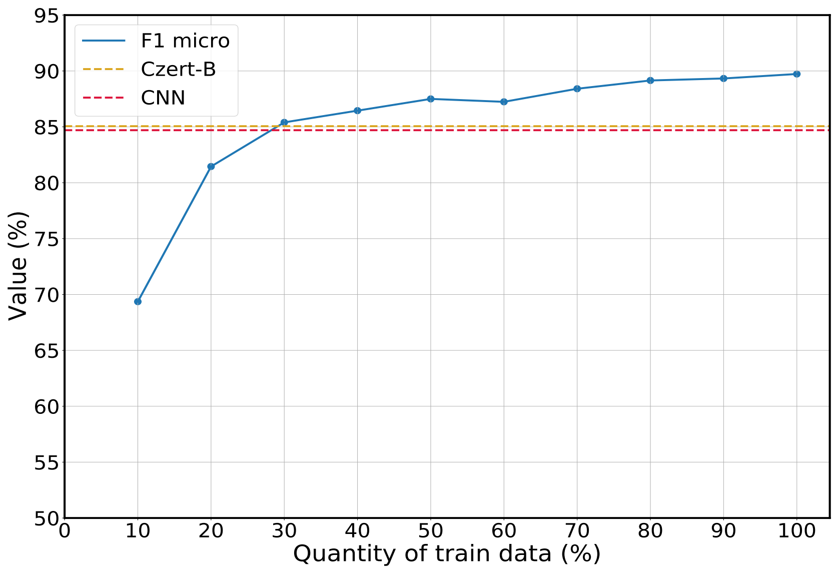


Figure 4.6: Graph of results with respect to the amount of training data.

4.3.3 Results on the Contracts Dataset

In this section, we focus on the evaluation of the contracts dataset that we created and presented in Section 2.3. Please note that this dataset has only an automatic annotation of the train and development sets. Because the development set is quite large, we use only 10% of the original development set for performance reasons. We are aware that this dataset contains a certain amount of data noise, which is why we trained our model in a sentiment analysis task even on random noisy data. That experiment showed that our model can handle quite a lot of noise in the data. For this reason, our currently used model and hyperparameters are based on the settings of the model that we used in the sentiment analysis task. We only changed the number of neurons in the output layer due to the different number of categories, and batch size, which we tried to use the maximum amount that GPU memory capacities allowed us. We also trained everything with batch size 8, which was the maximum amount which we were able to use in any case, for a correct comparison of the experiments. Each epoch consists of 20 000 randomly selected contracts for performance reasons. The used hyperparameters are shown in Table 4.9.

Batch size	8, 16, 32
Learning rate peak	1e-5
Dropout	0.1
Epochs	50
Contracts per epoch	20 000
Frozen epochs	1
Warmp-up	10%
Number of labels	22, 105

Table 4.9: Table of hyperparameters for fine-tuning.

This dataset is interesting because the individual contracts are much longer than the documents or comments classified by us in previous experiments. Therefore, we focused here on experiments about the selection of a specific part of contracts and the number of windows used in BERT. For this reason, we recorded the parts of the contracts, where the most keywords are located, during creation of the dataset. They are the mentioned *windows*, the ranges containing the largest number of keywords used during automatic classification.

The texts from contracts used in the experiments are as follows:

- metainfo (recipient, sender and subject of the contract),
- subject of the contract,

- first 400 tokens of the contract,
- first 500 tokens of the contract,
- first window,
- first two windows,
- all windows.

We remind that selected windows of the contracts are sorted according to the number of keywords they contain, so that the first window contains the most keywords. When using several windows at the same time during training, all windows are classified separately, and the result category is determined according to the average of the predictions of the individual windows.

The mentioned used parts of the contracts we also combined with each other. Specifically, the metainfo and the subject of the contract are shorter parts of the contracts, and contain brief and specific information from the contract. Therefore, we decided to perform experiments, in which we add them individually to the beginning of the already selected and mentioned parts of the contracts. This means, for example, that we add the subject to the beginning of each window, when we were using all windows. In the case we exceeded the limit of 512 tokens, we used only the first 512 tokens. The total number of configurations in terms of the use of contracts parts and batch size was 25, so we performed 25 different experiments. Their results are shown in Table 4.10, in which we also present the evaluation on main categories for each experiment. Subsequently, we were interested how big is the difference in evaluation on the main categories between the training on all 105 categories and just 22 main categories. For this reason, we performed five experiments, where our model was trained only on the main categories. The results are presented at the end of Table 4.10.

Text content	Accuracy				Batch size
	All categories		Main categories		
	Dev	Test	Dev	Test	
Metainfo	60.37	34.28	-	53.03	8
Subject	55.81	33.89	-	54.59	8
	52.18	36.04	-	54.88	16
	47.93	35.45	-	**61.23**	32
1. 400 tokens of text	63.49	35.16	-	52.25	8
	63.07	35.84	-	53.13	16
1. 400 of (subject + text)	64.63	**37.99**	-	55.96	8
	63.07	36.72	-	55.27	16
1. 400 of (metainfo + text)	65.66	**37.99**	-	56.45	8
	63.28	37.79	-	57.03	16
1. 500 tokens of text	64.83	36.13	-	51.95	8
1. 500 of (subject + text)	57.88	32.91	-	52.05	8
1. 500 of (metainfo + text)	63.80	36.33	-	55.08	8
1. window	66.29	34.57	-	52.25	8
	65.04	35.74	-	52.34	16
Subject + 1. window	67.12	37.60	-	55.37	8
Metainfo + 1. window	68.57	37.01	-	55.57	8
1-2. windows	46.58	25.20	-	35.64	8
	46.37	28.91	-	40.33	16
Subject + 1-2. windows	43.88	26.66	-	39.45	8
Metainfo + 1-2. windows	46.99	26.27	-	36.52	8
All windows	29.56	19.24	-	32.03	8
	33.09	22.27	-	35.74	16
Subject + all windows	31.85	19.43	-	28.22	8
Metainfo + all windows	34.44	19.63	-	27.73	8
Metainfo	-	-	76.97	54.79	8
Subject	-	-	65.77	**63.67**	32
1. 400 of (metainfo + text)	-	-	78.63	54.59	8
Metainfo + 1. window	-	-	81.33	56.93	8
Metainfo + all windows	-	-	47.51	41.31	8
Hlídač Státu	100	17.87	100	32.03	-

Table 4.10: Overall results of contracts.

Keep in mind that our goal is not to achieve 100% on the development set in interpreting the results as Hlídač Státu. In that case we would make the same classification errors as them. We do not know exactly how many contracts Hlídač Státu classifies incorrectly. Based on our experiment about the data noise in the sentiment analysis task (Section 4.3.1), we expected that we would be able to eliminate some amount of bad classification to a certain amount of data noise. However, it is important to remind that there we classified into three classes and all the errors in the data were made randomly.

Results on All Categories

The best results from training on all categories in terms of the text part of the contract we achieve on the first 400 tokens of contracts starting with the subject or metainfo of the contract. Specifically, the best achieved value is 37.99%, see Table 4.10. However, it is unfair to compare with Hlídač Státu on the test set, because there are mainly those contracts that they classified incorrectly, and in some sense this type of contracts was difficult for them. We calculated the Pearson's and Spearman's correlation coefficients for better interpretation of the results on the development set, they are shown in Table 4.11. Pearson's correlation coefficient describes a linear relation between two values, Spearman's corresponds to a non-linear correlation, which only needs to be monotonous. The correlations between development and test results in training on all categories are very high in both cases. Pearson's correlation coefficient is 0.94, which is considered as a very high dependence. Figure 4.7 shows a scatter plot that indicates the relation between development and test results, and shows a positive correlation, which means that both variables tend to increase in correlation. Based on these correlations, we can trust more the results on the development set and thus the better the value on the development set, the better the overall model. The best result on the development set is 68.57% and the corresponding value on the test set is 37.01%, which is also almost the best result, and this further convinces us that the correlation makes sense.

Type of correlation	**Correlation**	
	All categories	**Main categories**
Pearson's	0.94	0.64
Spearman's	0.84	0.30

Table 4.11: Table of corellations with Pearson and Spearman correlation coefficients.

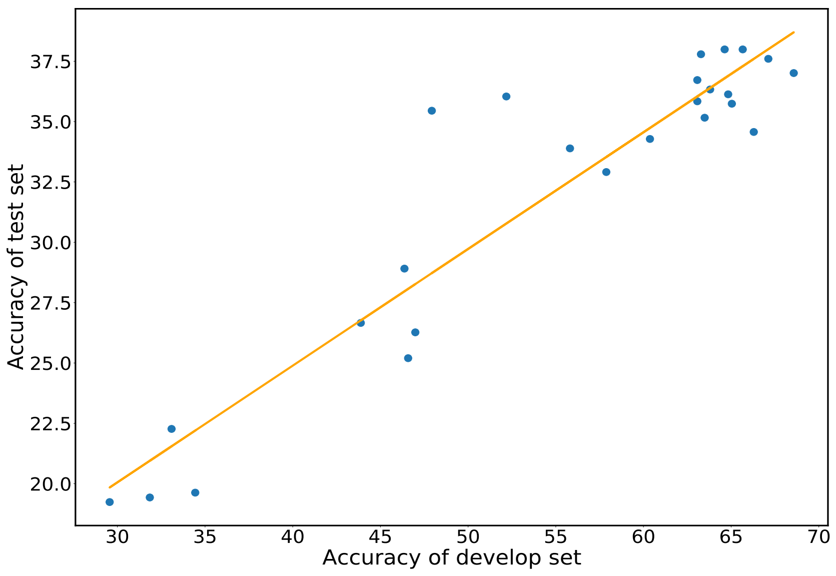


Figure 4.7: Scatter plot indicating the relation between the accuracy development set and the test set on all categories. It can be seen from the graph that there is a positive correlation between these variables. The relation between these variables seems to be linear.

Results on Main Categories

We saw from the Facebook dataset results that the coincidence can be removed quite well by our model. In this case, however, we most likely have systematic errors in the contract classification. It relates also with the method of using keywords for classifying contracts used by Hlídač Státu. Our model could not deal with these systematic errors easily and learned to classify according to that error as well. To indicate that there are systematic errors in the date, consider the achieved results using the *Subject* of 35.45% (classification of all categories) and if the results are evaluated on the main categories, we get 61.23% and thus the difference is almost 26%. When we take the results obtained with *Metainfo + 1. window* 37.01% (classification of all categories) and its corresponding evaluation on the main categories 55.57%, the difference is 18.56%. This suggests that if we overfitted on some subcategory, we would have chosen the right main category anyway, but this did not happen, so the systematic errors are more likely at the level of main categories.

Overfitting is best seen with the text *Metainfo + 1. window* in main categories, where we reached 81.33% on development, which is the greatest value we achieved, see Table 4.10. If we did not have the test set, we would choose this model. However, the model reaches only 56.93% on the test set, which is not the best performance in main categories. It illustrates that the result on the development set in training on main categories does not correlate as strongly with the results on the test set as in the previous case.

The fact that we achieve the best results on the main categories using the text part of contracts with only the subject (61.23% and 63.67%) also indicates that the original classification has systematic errors. With such a small amount of text as the subject of the contract, our model is weaker and cannot remember the mistakes made by their classification system because those mistakes cannot be observed just from the subject. If our model sees more text from the contract, can also learn to make mistakes, and then make them.

If we look at the correlations of results trained on main categories, we can see in Table 4.11 that Pearson's coefficient is 0.64 and Spearman's is only 0.3, which are considered to be medium to slight dependence. We achieved the best result of 63.67% but its development value (65.77%) is the second worst of 5 measurements. Figure 4.8 shows a scatter plot of the relation between results on the development set and the test set showing a positive correlation, but compared to Figure 4.7, less correlation of the result can be seen.

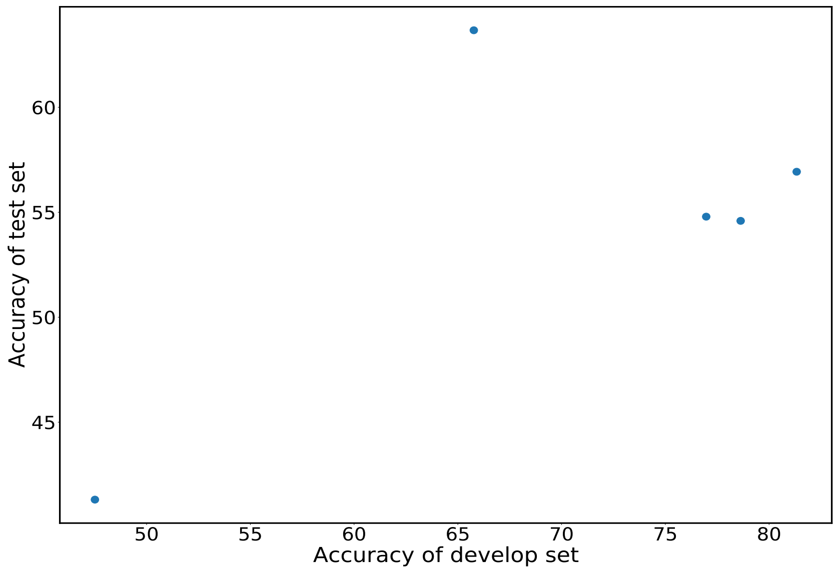


Figure 4.8: Scatter plot indicating the relation between the accuracy development set and the test set on main categories.

Confusion Matrix of the Main Categories Prediction

We show the confusion matrix for the best evaluation for the main categories in Figure 4.9. We achieve 0% in two categories for two different reasons. In the first case, it is the category *0 – Other*, which is represented in the training set with only one contract, and therefore our model could not learn to classify this category. In the second case, it is the category *10800 – Natural resources*, which has no representation in the test set. It also follows from the fact that it has 0% values in the whole row of the matrix. Categories that our model classifies the best are *11600 – Technical services* and *10900 – Energy*. Both achieve almost

90%. The biggest misclassification of one category into another occurs in the category *11700 – Science, research and development*, where we classify 1/3 of the contracts into the category *10000 – IT*. Furthermore, we misclassify mostly the categories *11800 – Advertising and marketing services* and *12000 – Donations and subsidies*. Both we classify evenly into several other categories.

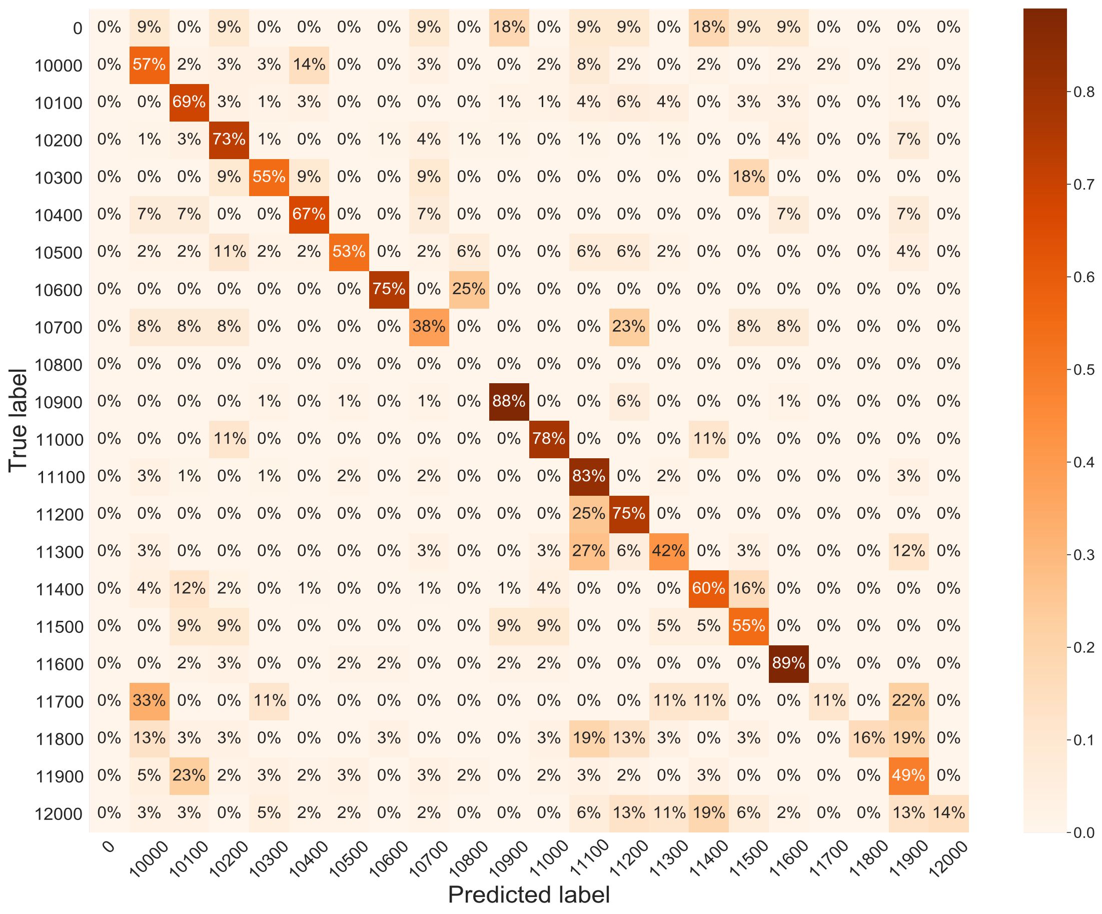


Figure 4.9: Confusion matrix of main categories (text content: subject).

Experiments with the Amount of Contract Texts

Selecting the text using windows is beneficial only if we use the first window. It can be seen in cases of selecting the *(Subject or Metainfo) + 1st window*. If we compare the results obtained from the *First 400 tokens (of Subject or Metainfo + text)*, we see that the results on the development set are slightly better. A small part of this improvement may be due to the fact that the first window tends to be at the beginning of the contract, and then these two choices overlap. Furthermore, the first window often has much more keywords compared to the other windows in the contract. This relates to the fact that the rest of the contract is more general. For this reason, using multiple windows at once does not work properly, as they are often too general and make the overall classification average worse. We can see that from the results in a gradual deterioration when using more and more windows. The worst results are achieved with the use of all windows, and therefore, we recommend using only the first window for classification or possibly using only the subject itself.

Selecting the batch size

For the batch size, we noticed an improvement with a larger choice only in evaluation of main categories. The best result are achieved with batch size 32. The batch size was not that important when classifying into all categories. The best results of classification into all categories was achieved with batch size 8.

Manual Evaluation Main Categories on the Development Set

Out of curiosity, we performed a manual evaluation of the main categories on the development set, specifically on 100 contracts. We compared the predictions on the development set obtained by our model using the Subject as a text content of the contract (we achieved the best result on the main categories with this configuration) and the predictions of the Hlídač Státu. The prediction successes with the corresponding standard deviations obtained using a bootstrap resampling with 100 000 samples are shown in Table 4.12. The contract sometimes belong to several categories or the categories may overlap, so it was difficult to determine whether the predictions were right. In such cases, we accepted both categories.

Prediction	**Result**	**Standard deviation**
Subject	69	± 4.63
Hlídač Státu	62	± 4.86

Table 4.12: The success of predictions on the development set with the corresponding standard deviations.

Conclusion

The goal of this thesis was to evaluate the text classification in low resource settings, and to analyze the accuracy of the model depending on the size and quality of training data. When we were choosing the datasets, we focused on Czech texts. We used our model, which uses the pre-trained RobeCzech model, for all measurements and experiments.

We measured the performance of the model depending on the quality of training data on the Facebook dataset, where we found that our model handles quite well 20% of noisy data with a decrease in performance of less than 3%. Furthermore, up to the noise of 45% of training data, we achieved stable results above 71%, with which we still surpass the results obtained with methods such as SVM or Maximum entropy. A big drop in performance occurred after 60% of the data was noisy. When examining the size of the training data needed to achieve good results, we found that we only need to take 4% of the training set, and train the task on it to overcome the results obtained by older methods. To compare with more modern techniques such as Czert, we need to take only 10% of the training data and we get better results.

Another dataset, on which we analyzed the required size of training data during training, was Czech Text Document Corpus. We evaluated the multi-label classification on this dataset. We found that the larger decrease in performance occurred only when training with 20% or less data, and until then, the overall evaluation decreased only slightly. Until we used less than 30% of the training data, we achieved better results than have been published so far, including those obtained with modern techniques. Our result obtained on the whole set of training data is 89.46%, and to our knowledge, we set a new state-of-the-art result on this dataset.

The last dataset, on which we evaluated the text classification, focuses on long texts of contracts and was partially created by us. We classified contracts in this dataset into 105 categories or into 22 main categories. In both cases, we tried to understand the obtained results, and we experimented with choosing the right part and size of the text of contracts. We have found that for the classification of main categories is better to take only the subject of the contract without additional text, which we assume to prevent copying of systematic errors from the automatic annotation of contracts. When classifying into all categories, the advantage of the window selection of the text of the contract created by us partially occured, but we achieved the best results with the selection of the text, where we used the beginning of the contract with the subject or metainfo.

Future work

One of the benefits of this work is the creation of a new dataset of Czech contracts, which was non-public up to now, so it would be useful to continue with the complex research of possibilities of choosing the right part of contracts for classification. We would recommend to try different shifts of the obtained windows or to create new windows with different sizes, because the window size was set just to 300 tokens in our case. Last but not least, the obtained windows could be tested in different positions. It would be also interesting to try training on the obtained predictions, so-called self-training, to partially eliminate the systematic errors.

Bibliography

Czes, 2011. URL `http://hdl.handle.net/11858/00-097C-0000-0001-CCCF-C`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

George Bebis and Michael Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31, 1994.

Daniel Berrar. Cross-validation. *Encyclopedia of bioinformatics and computational biology*, 1:542–545, 2019.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Meng Joo Er, Rajasekar Venkatesan, and Ning Wang. An online universal classifier for binary, multi-class and multi-label classification. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 003701–003706. IEEE, 2016.

César Ferri, José Hernández-Orallo, and R Modroiu. An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, 30(1): 27–38, 2009.

Kunihiko Fukushima. Neocognitron for handwritten digit recognition. *Neurocomputing*, 51:161–180, 2003.

Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, 2019.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. Technical report, Carnegie-Mellon Univ Pittsburgh Pa School of Computer Science, 2010.

Ivan Habernal, Tomáš Ptáček, and Josef Steinberger. Facebook data for sentiment analysis, 2013a. URL `http://hdl.handle.net/11858/00-097C-0000-0022-FE82-7`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Ivan Habernal, Tomáš Ptáček, and Josef Steinberger. Sentiment analysis in czech social media using supervised machine learning. In *Proceedings of the 4th workshop on computational approaches to subjectivity, sentiment and social media analysis*, pages 65–74, 2013b.

Jan Hajic, Jarmila Panevová, Eva Hajicová, Petr Sgall, Petr Pajas, Jan Štepánek, Jiří Havelka, Marie Mikulová, Zdenek Zabokrtskỳ, Magda Ševcıková-Razımová, et al. Prague dependency treebank 2.0. *CD-ROM, Linguistic Data Consortium, LDC Catalog No.: LDC2006T01, Philadelphia*, 98, 2006.

Peter Harremoës and Flemming Topsøe. Maximum entropy fundamentals. *Entropy*, 3(3):191–226, 2001.

Michal Hrala and Pavel Král. Evaluation of the document classification approaches. In *Proceedings of the 8th International Conference on Computer Recognition Systems CORES 2013*, pages 877–885. Springer, 2013.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014. URL `http://arxiv.org/abs/1408.5882`.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Daniel Kondratyuk, Tomáš Gavenčiak, Milan Straka, and Jan Hajič. Lemmatag: Jointly tagging and lemmatizing for morphologically-rich languages with brnns. *arXiv preprint arXiv:1808.03703*, 2018.

Pavel Král and Ladislav Lenc. Czech text document corpus v 2.0. *arXiv preprint arXiv:1710.02365*, 2017.

Michal Křen, Václav Cvrček, Tomáš Čapka, Anna Čermáková, Milena Hnátková, Lucie Chlumská, Tomáš Jelínek, Dominika Kováříková, Vladimír Petkevič, Pavel Procházka, Hana Skoumalová, Michal Škrabal, Petr Truneček, Pavel Vondřička, and Adrian Zasina. SYN v4: large corpus of written czech, 2016. URL `http://hdl.handle.net/11234/1-1846`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Gustavo Laboreiro, Luís Sarmento, Jorge Teixeira, and Eugénio Oliveira. Tokenizing micro-blogging messages using a text classification approach. In *Proceedings of the fourth workshop on Analytics for noisy unstructured text data*, pages 81–88, 2010.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Ladislav Lenc and Pavel Král. Deep neural networks for czech multi-label document classification. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 460–471. Springer, 2016.

Patrick Lewis, Barlas Oğuz, Ruty Rinott, Sebastian Riedel, and Holger Schwenk. Mlqa: Evaluating cross-lingual extractive question answering. *arXiv preprint arXiv:1910.07475*, 2019.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019a.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019b.

Martin Majliš. W2C – web to corpus – corpora, 2011. URL `http://hdl.handle.net/11858/00-097C-0000-0022-6133-9`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Jakub Maroušek. Automatická klasifikace smluv pro portál hlidacsmluv.cz. Master's thesis, Univerzita Karlova, Matematicko-fyzikální fakulta, Katedra softwarového inženýrství, Praha, 2020.

Fionn Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5-6):183–197, 1991.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

Peltarion. Learning rate schedule, 2021. URL `https://peltarion.com/knowledge-center/documentation/modeling-view/run-a-model/optimization-principles-(in-deep-learning)/learning-rate-schedule`.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. *Technical report, OpenAI*, 2018.

Jakub Sido, Ondřej Pražák, Pavel Přibáň, Jan Pašek, Michal Seják, and Miloslav Konopík. Czert–czech bert-like model for language representation. *arXiv preprint arXiv:2103.13031*, 2021.

Milan Straka, Jakub Náplava, Jana Straková, and David Samuel. Robeczech: Czech roberta, a monolingual contextualized language representation model. *CoRR*, abs/2105.11314, 2021. URL `https://arxiv.org/abs/2105.11314`.

Jana Straková, Milan Straka, and Jan Hajič. Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 13–18, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P14/P14-5003.pdf`.

G Tsoumakas and I Katakis. Multi-label classification: An overview, dept. of informatics. *Aristotle University of Thessaloniki, Greece*, 2006.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

B Vijaykumar and Trilochan Vikramkumar. Bayes and naive-bayes classifier. *Computer Science & Engineering. Rajiv Gandhi University of Knowledge Technologies Andhra Pradesh, India*, 2014.

Hanna M Wallach. Topic modeling: beyond bag-of-words. In *Proceedings of the 23rd international conference on Machine learning*, pages 977–984, 2006.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

List of Figures

List of Tables

List of Abbreviations

AIC Artificial Intelligence Cluster. 43

ALBERT A Lite BERT. 15, 37

BERT Bidirectional Encoder Representations from Transformers. iii, 3, 5, 10, 11, 12, 13, 14, 15, 19, 20, 23, 26, 35, 37, 38, 39, 44, 49, 50, 52, 67

BoW Bag of Words. 36

BPE Byte-Pair Encoding. 15

CDF Cumulative Distribution Function. 32

CNN Convolutional Neural Network. 36

FDNN Feed-forward Deep Neural Network. 36, 37

FFN Feed Forward Network. 6, 8, 15

GELU Gaussian Error Linear Units. 11

GLUE General Language Understand Evaluation. 44

GPU graphics processing unit. 43, 44, 49, 52

mBERT multilingual BERT. 14

MLP Multi-layer Perceptron. 36

MLQA MultiLingual Question Answering. 14

MorphoDiTa Morphological Dictionary and Tagger. 24, 25

NLP natural language processing. 3, 36, 43

PDT Prague Dependency Treebank. 35

PE Positional Encoding. 8, 9

ReLU Rectified Linear Unit. 8, 11

RNN Recurrent Neural Network. 5, 8, 9, 24

RoBERTa Robustly optimized BERT approach. 14, 15, 35, 38, 43, 44

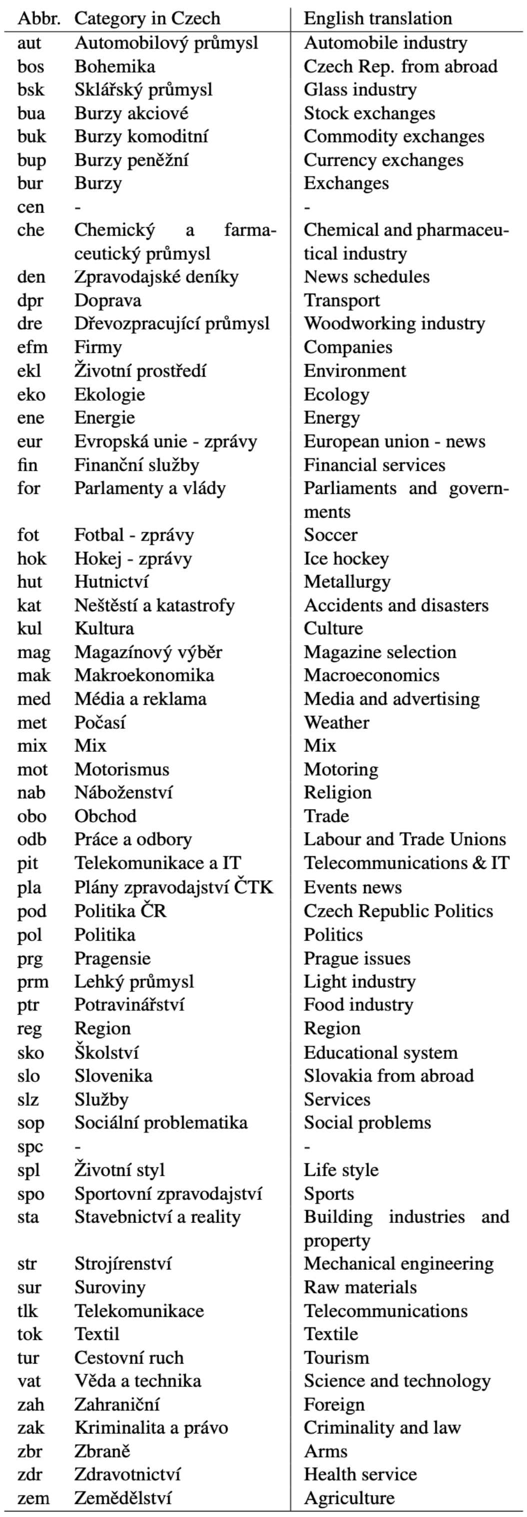
SQuAD Stanford Question Answering Dataset. 13, 44, 67

SVM Support Vector Machines. 3, 4, 36, 47, 60

ÚFAL Institute of Formal and Applied Linguistics. 23, 24, 43

A. Attachments

A.1 List of the Categories



Abbr.	Category in Czech	English translation
aut	Automobilový průmysl	Automobile industry
bos	Bohemika	Czech Rep. from abroad
bsk	Sklářský průmysl	Glass industry
bua	Burzy akciové	Stock exchanges
buk	Burzy komoditní	Commodity exchanges
bup	Burzy peněžní	Currency exchanges
bur	Burzy	Exchanges
cen	-	-
che	Chemický a farmaceutický průmysl	Chemical and pharmaceutical industry
den	Zpravodajské deníky	News schedules
dpr	Doprava	Transport
dre	Dřevozpracující průmysl	Woodworking industry
efm	Firmy	Companies
ekl	Životní prostředí	Environment
eko	Ekologie	Ecology
ene	Energie	Energy
eur	Evropská unie - zprávy	European union - news
fin	Finanční služby	Financial services
for	Parlamenty a vlády	Parliaments and governments
fot	Fotbal - zprávy	Soccer
hok	Hokej - zprávy	Ice hockey
hut	Hutnictví	Metallurgy
kat	Neštěstí a katastrofy	Accidents and disasters
kul	Kultura	Culture
mag	Magazínový výběr	Magazine selection
mak	Makroekonomika	Macroeconomics
med	Média a reklama	Media and advertising
met	Počasí	Weather
mix	Mix	Mix
mot	Motorismus	Motoring
nab	Náboženství	Religion
obo	Obchod	Trade
odb	Práce a odbory	Labour and Trade Unions
pit	Telekomunikace a IT	Telecommunications & IT
pla	Plány zpravodajství ČTK	Events news
pod	Politika ČR	Czech Republic Politics
pol	Politika	Politics
prg	Pragensie	Prague issues
prm	Lehký průmysl	Light industry
ptr	Potravinářství	Food industry
reg	Region	Region
sko	Školství	Educational system
slo	Slovenika	Slovakia from abroad
slz	Služby	Services
sop	Sociální problematika	Social problems
spc	-	-
spl	Životní styl	Life style
spo	Sportovní zpravodajství	Sports
sta	Stavebnictví a reality	Building industries and property
str	Strojírenství	Mechanical engineering
sur	Suroviny	Raw materials
tlk	Telekomunikace	Telecommunications
tok	Textil	Textile
tur	Cestovní ruch	Tourism
vat	Věda a technika	Science and technology
zah	Zahraniční	Foreign
zak	Kriminalita a právo	Criminality and law
zbr	Zbraně	Arms
zdr	Zdravotnictví	Health service
zem	Zemědělství	Agriculture

Figure A.1: List of the categories, from paper Král and Lenc [2017], Table 3.

A.2 Description of Scripts Used for Creating the Dataset of Contracts.

The scripts are presented in the order in which was creating the dataset.

- **categoriesCZ.json** – Json file with all labels and names of the categories.
- **notations_of_files.txt** – Name list of the categories used for naming the obtained contracts.
- **get_contracts.py** – Obtaining contracts according to their relevance in the categories from the Hlídač Státu portal.
- **all_keywords_uniq.txt** – List of keywords used for obtaining windows from the text of the contract.
- **robeczech_tokenizer.py** – Tokenizer used for obtaining the subwords.
- **lemmatizer.py** – Script for obtaining the lemmas from the text and for creating the n-grams from obtained lemmas.
- **json_parser.py** – Parsing the .json contract record. It allows us to return content of used objects, such as subject, recipient and the text of the contract.
- **windows.py** – Obtaining individual windows from the text of the contract with the content of as many keywords as possible. Keywords are centralized in the center of the window.
- **create_pre_final_dataset.py** – Preparation of the contracts into the final form by adding the label, plaintext of the contract and windows to each contract.
- **final_contracts_dataset.sh** – Script for AIC cluster used for automatic preparation of final contracts.
- **merged_json_files.sh** – Merging all acquired contracts into one .jsonl file.
- **create_sets_for_training.py** – Creation of the training and development sets.

A.3 Description of Scripts for Creating Training Models and Experiments

Facebook Dataset

- **sentiment_analysis_model.py** – Script for training the model.
- **text_classification_dataset.py** – Preparation of data and loading data for training.
- **robeczech_tokenizer.py** – Tokenizer used for obtaining the subwords.
- **create_amount_of_train_data.py** – Creation of training data with different amounts of posts from the original training set.
- **create_noisy_data.py** – Creation of training data with different amounts of data noise.
- **evaluate_predict_file.py** – Calculation of the evaluation from the obtained predict file.

Czech Text Document Corpus

- **cz_corpus_model.py** – Script for training the model.
- **cz_corpus_text_classification.py** – Preparation of data and loading data for training.
- **robeczech_tokenizer.py** – Tokenizer used for obtaining the subwords.
- **create_amount_of_train_data.py** – Creation of training data with different amounts of documents from the original training set.
- **merged_txt_files.py** – Merging individual text files of documents into one text file.
- **preprocessing_dataset.py** – Preparation of the acquired text file for training. Reduction of the documents to a length of 300 tokens. Creating folds for training.

Contracts Dataset

- **contracts_classification_model.py** – Script for training the model.
- **contracts_classification_dataset.py** – Preparation of data and loading data for training.

- **evaluate_and_confusion_matrix.py** – Evaluation of the predicted file and obtaining the confusion matrix for main categories.