



Assignment of bachelor's thesis

Title:	Named entity recognition for poetic texts
Student:	Ondřej Černý
Supervisor:	Ing. Karel Klouda, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2023/2024

Instructions

This work is part of a cooperation with the Institute of Czech Literature (ICL), which involves processing more than 1,300 digitized poetry collections from the 19th and early 20th centuries. The aim is to identify named entities in these poetic texts using NLP techniques.

- 1) Explore the data of the Corpus of Czech Verse [1].
- 2) Explore and survey NLP techniques for named entity recognition (NER).
- 3) Find or create a suitable dataset for training a NER model: at least entities representing a person or a location should be reliably labeled and distinguished in the dataset.
- 4) Apply the selected method to the data from the corpus, and evaluate the results.

Bachelor's thesis

NAMED ENTITY RECOGNITION FOR POETIC TEXTS

Ondřej Černý

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Karel Klouda, Ph.D.
May 11, 2023

Czech Technical University in Prague

Faculty of Information Technology

© 2023 Ondřej Černý. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis: Černý Ondřej. *Named entity recognition for poetic texts.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Contents

Acknowledgments	vii
Declaration	viii
Abstract	ix
List of abbreviations	x
Introduction	1
1 Named Entity Recognition	3
1.1 NER in Czech poetry	3
1.1.1 Entities in Czech poetry	4
1.2 Ways to solve NER	4
1.2.1 Maximum Entropy Markov Model (MEM) for NER	4
1.2.2 BERT for NER	5
1.3 Usage of NER	5
1.3.1 Usage of NER on Czech Poetry	5
1.4 Entity labeling	5
1.4.1 BIO tagging	6
2 Corpus of Czech Verse	7
2.1 Corpus Scheme	7
2.1.1 Poem Schema	7
2.1.2 Line Schema	7
2.2 Prague positional system	8
3 Entity selection	11
3.1 Metrics used for evaluation of the rule-based approach	11
3.2 Capital words	12
3.3 Single-word entities	12
3.3.1 Capital lemmas x Capital tokens	12
3.3.2 Using Morphology	12
3.3.3 Forbidden Lemmas	13
3.3.4 Final version	13
3.4 Multi-word entities	13
3.4.1 Czech grammar on multi-word entities	13
3.4.2 Position of words in entity	14
3.5 Final version of rule-based approach	14
3.5.1 Selecting potential entities	17
3.6 Result of entity selection	18

4	Categorization of Entities	21
4.1	Categories description	21
4.2	Implementation of categorization	21
4.2.1	Wikipedia page layout	22
4.2.2	Categorizing entities using its Wikipedia page	22
4.2.3	Wikidata layout	24
4.2.4	Selecting entities using Wikidata	24
4.2.5	Categorizing entities of category Other	25
4.3	Result of categorization	26
5	Theoretical description of our model	27
5.1	Recurrent neural network (RNN)	27
5.2	BiLSTM-CRF	28
5.2.1	Long Short-Term Memory networks (LSTM)	28
5.2.2	Bidirectional LSTM Network (BiLSTM)	29
5.2.3	Conditional Random Fields (CRF)	30
5.2.4	BiLSTM-CRF	30
5.2.5	Theory of training BiLSTM-CRF network	31
6	Implementation of our model	33
6.1	Training data	33
6.1.1	Stanzas vs. Poems dataset	34
6.1.2	Removing stanzas with potentially unmarked entities	35
6.1.3	Train, Dev and Test split	35
6.1.4	Input format for training data	35
6.2	Metrics used for evaluation	36
6.3	Removing Real Person as an entity category	36
6.4	Hyperparameter testing/tuning	39
6.4.1	Word embeddings	39
6.4.2	Input features	41
6.4.3	Embedding dimension size for morphological tag	42
6.4.4	Hyperparameter tuning	44
6.5	Final model	45
6.5.1	Results of final model	47
6.5.2	Saving and visualizing results	47
7	Conclusion	51
7.1	Room for improvement	51
	Source code	55

List of Figures

1.1	Example of NER [2]	3
1.2	BIO tagging [8]	6
2.1	Poem Schema [12]	9
2.2	Example of morphological tag from CCV dataset (16 positions)	10
2.3	Tag Structure as found in [13] (15 positions)	10
4.1	Top of Wikipedia page	22
4.2	Bottom of Wikipedia page	23
4.3	Search in token format	23
4.4	Search in lemma format	23
4.5	Keywords used for categorization with Wikipedia page categories	24
4.6	Wikipedia category example	24
4.7	Wikidata page layout	25
4.8	Keywords used for categorization with Wikidata	25
4.9	Ratios of categorized entities	26
5.1	RNN example [23]	28
5.2	Memory cell [23]	29
5.3	LSTM network example [23]	29
5.4	BiLSTM network example [23]	30
5.5	CRF network example [23]	30
5.6	BiLSTM-CRF network example [23]	31
5.7	Detailed BiLSTM-CRF network example [20]	32
5.8	BiLSTM-CRF network: Training procedure [23]	32
6.1	Poems vs. Stanzas dataset: Entity ratios	34
6.2	Example of input text format	36
6.3	Comparison between models with or without Real Person category	37
6.4	Lemma vs. Token embedding	40
6.5	Input features (zoomed y axis so the difference between F1 scores can be seen better)	42
6.6	Different morphological tag sizes of embedding (zoomed y axis so the difference between F1 scores can be seen better)	44
6.7	Final model F1 score (Dev and Test datasets)	46
6.8	Entity categories distribution	47
6.9	Resulting schema example	48
6.10	Visualization example	49

List of Tables

3.1	Entity selection overview	18
6.1	Set of valid stanzas: Number of stanzas for this set out of the total number of stanzas in the CCV dataset	33
6.2	Poems vs. Stanzas dataset: Number of entities	34
6.3	Confusion matrix example	36
6.4	Confusion matrix model without Real Person entity	38
6.5	Confusion matrix model with Real Person entity	38
6.6	Confusion matrix for the model with lemma embedding	40
6.7	Confusion matrix for the model with token embedding	41
6.8	Confusion matrix for the model with pre-trained lemma embedding and morphological tag feature	43
6.9	Confusion matrix for the model with pre-trained lemma and token embedding and morphological tag feature	43
6.10	Best hyperparameter values	45
6.11	Confusion matrix for the final model created on test dataset	46
6.12	Number of entities in the whole CCV	47

List of code listings

3.1	Selecting potential entities	19
-----	--	----

Firstly, I would like to sincerely thank my supervisor, Ing. Karel Klouda, Ph.D., for good guidance and great advice and for making the writing of this thesis fun and enjoyable. I am also grateful for the support from my family and friends not only in my studies but also generally in life. And lastly, I am much obliged to the Faculty of Information Technology at CTU and also to the Czech Ski and Snowboard Federation for allowing me to combine the life of a professional athlete with the life of a university student.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 11, 2023

.....

Abstract

The result of this work is a program that uses Natural Language Processing (NLP) techniques to identify named entities in the Corpus of Czech Verse (CCV). It is part of a cooperation with the Institute of Czech Literature (ICL). Since CCV is not even partially labeled for entity recognition, we first create a set of rules, and using those, we select entities from the poems. These entities are later on categorized into different entity categories using data from Wikipedia. After that, these categorized entities are used as training data for a BiLSTM-CRF neural network that is trained and fine-tuned for NER on the CCV. The resulting model can find and distinguish entities of Place, Person, Mystic Person, and Other. Since the text in the CCV is not labeled for NER, we cannot know the exact accuracy of the final BiLSTM-CRF model. If we would consider the data that are used for training of this model to be 100% accurate, then the final model would have achieved an accuracy of 0.99904 and an F1 score of 0.9532.

Keywords BiLSTM-CRF, Corpus of Czech Verse, Lemmatization, Morphology, Natural language processing, Recurrent neural network, Named entity recognition, Word2Vec

Abstrakt

Výsledkem této práce je program, který využívá techniky Zpracování přirozeného jazyka k identifikaci pojmenovaných entit v Korpusu českého verše (KČV). Jedná se o součást spolupráce s Ústavem pro českou literaturu. Jelikož KČV není ani z části označen pro rozpoznávání pojmenovaných entit (RPE), musíme zprvu vytvořit množinu pravidel, se kterými najdeme entity v textu. Tyto entity jsou následně kategorizovány s pomocí dat z Wikipedie. Poté jsou tyto kategorizované entity využity jakožto trénovací data pro BiLSTM-CRF neuronovou síť, která je následně trénována a vyladěna pro RPE na KČV. Výsledný model je schopen nalézt a rozlišit entity místa, osob, mystických osob a jiné. Jelikož text v KČV není označen pro RPE nejsme schopni udát skutečnou přesnost finálního BiLSTM-CRF modelu. Pokud bychom počítali s tím, že trénovací data použita na natrénování tohoto modelu jsou 100% přesná, pak by výsledný model dosáhl přesnosti 0.99904 a F1 skóre 0.9532.

Klíčová slova BiLSTM-CRF, Korpus českého verše, Lemmatizace, Morfologie, Rekurentní neuronová síť, Rozpoznávání pojmenovaných entit, Word2Vec, Zpracování přirozeného jazyka

List of abbreviations

BERT	Bidirectional Encoder Representations from Transformers
BIO	Beginning, Inside and Outside
BiLSTM	Bidirectional LSTM
BiLSTM-CRF	Bidirectional LSTM network with CRF layer
CNN	Convolutional neural network
CRF	Conditional random fields
LSTM	Long short-term memory
MEMM	Maximum-entropy Markov model
NER	Named entity recognition
NLP	Natural language processing
POS	Part of speech
RNN	Recurrent neural network

Introduction

Named Entity Recognition (NER) is a natural language processing task that involves identifying and extracting entities from text. In the context of poetry, NER can be used to identify and extract entities such as people and places from poems. By performing NER on poetry, we can gain insights into the themes, settings, and characters present in the poem. Even though there are currently many available models that can solve NER, not many of them are trained on the Czech language, and even fewer (if any) are trained on Czech poetry.

A model trained on poetry can differ from a NER model trained on plain text in a few ways. The language used in poetry can be more complex and stylized than everyday language since the poets often use words in a more creative and unusual way than we do in our daily lives. This can make it challenging for a NER model trained on plain text to accurately recognize named entities in poetry. Additionally, poetry often employs non-standard syntax and grammar, which can further complicate the task of entity recognition.

NER typically requires supervised learning. This means that a labeled dataset is used to train the NER model. Since the dataset (Corpus of Czech Verse [1]) that this thesis will be working with is not labeled in this manner, there is a need to label at least part of it to create training data for the model. Due to the lack of time and human resources (one student and one supervisor), the labeling cannot be done manually. Therefore, a rule-based approach will be created to label at least some of the data. So as a whole, this method of solving NER will not be supervised or unsupervised but rather something in between.

Motivation

The result of this thesis is supposed to be used by the Institute of Czech Literature by adding another layer of context to their Corpus of Czech Verse [1]. Furthermore, this work will produce one of the few (if not the only) NER model (BiLSTM-CRF model, to be exact) trained and explicitly fine-tuned for Czech poetry.

Thesis structure

This thesis begins with the theory surrounding NER in general and some possible ways how to solve it (Chapter 1). Then it describes the Corpus of Czech Verse (Chapter 2) and its structure. After that, it carries on with the creation of training data for the BiLSTM-CRF model (Chapter

3 and 4). Then it carries on with a theoretical description of the model used in this thesis (Chapter 5) and, after that, with implementation and training of said model (Chapter 6). Then the thesis finishes with a presentation of the results and a conclusion (Chapter 7).

Objectives

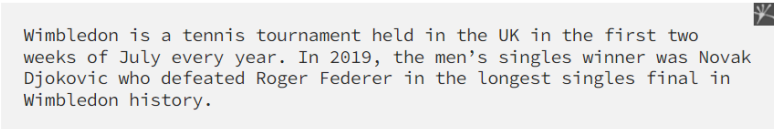
The objective of this thesis is to create a model that can identify named entities in the CCV. To do that, a training dataset will be created using a rule-based approach. Then this training data will be used to train BiLSTM-CRF neural network, which will be trained to achieve the best result possible. All of these steps and how they are implemented will be explained to the reader, including some theoretical background behind them.

Named Entity Recognition

This chapter familiarizes the reader with named entity recognition (NER), its uses, and ways how to solve it.

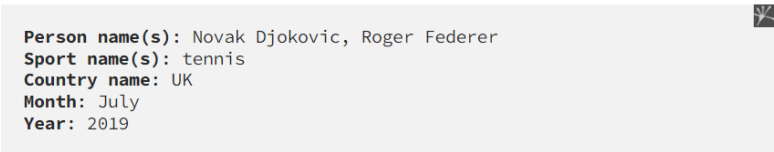
In natural language processing (NLP), named entity recognition (NER) is the problem of recognizing and extracting specific types of entities in text, such as people or place names. In fact, any concrete “thing” that has a name at any level of specificity. [2] As an example, we can look at Figure 1.1.

From



Wimbledon is a tennis tournament held in the UK in the first two weeks of July every year. In 2019, the men’s singles winner was Novak Djokovic who defeated Roger Federer in the longest singles final in Wimbledon history.

we might extract entities



Person name(s): Novak Djokovic, Roger Federer
Sport name(s): tennis
Country name: UK
Month: July
Year: 2019

■ **Figure 1.1** Example of NER [2]

1.1 NER in Czech poetry

NER in and of itself is quite a well-known problem in the NLP realm. There are lots of free-to-use models that can do NER. The problem is that most of these models are trained on English text, which is dominant on the Internet (Wikipedia, for example). Our dataset is not only

not in English but in Czech, which is not even in the same branch of languages as English (lexicographically speaking). There is also another problem, which is the fact that poetry is usually written in quite a specific way which is hard to come by in any other writing form. For example, the word order in poetry can sometimes be quite different from the word order used in regular text. This is usually due to the fact that poetry aims more than any other form of text to sound phonetically pleasing to the reader (use of rhymes, lots of metaphors, etc.).

1.1.1 Entities in Czech poetry

In poetry, the entities that we will encounter the most are usually entities of place and people. Especially in the dataset that is being used here (more about the dataset in Chapter 2), it is extremely rare to find, for example, names of companies as entities.

The entities that we decided to consider for our dataset are Person, Place, Real Person, Mystic Person, and Other. The reason is that any other entities did not seem to appear in the dataset in large enough quantities to create a category for them.

1.2 Ways to solve NER

Named Entity Recognition (NER) can be solved using various approaches, including dictionary-based, rule-based, and machine-learning methods.

One of the simplest ones is a dictionary-based approach. We have a dictionary of values for every entity type to be recognized. To recognize and extract the entities, we simply scan the text and find hits in the various dictionaries. A hit also reveals the entity type, as we know the dictionary that was hit. [2]

A rule-based approach involves defining a set of rules that can identify named entities. “In this method, a predefined set of rules for information extraction is used which are pattern-based and context-based. Pattern-based rule uses the morphological pattern of the words while context-based uses the context of the word given in the text document.” [3]

Unlike the previous two approaches, many machine learning models for solving NER require labeled training data (they are supervised models). We could separate these approaches into two further categories, and those are statistical-based (MEMM, CRF, etc.) and deep learning-based (LSTM, BERT, etc.) models.

In our approach, we will be using sort of a hybrid approach, where we will first apply the rule-based approach to our dataset (Chapter 2) and then use the result we get from that approach as training data for a Bidirectional Long Short Term Memory neural network (BiLSTM) that works together with CRF (more about those models later in Chapter 5).

1.2.1 Maximum Entropy Markov Model (MEMM) for NER

MEMM algorithm is based on a probabilistic model that takes into account the previous words in a sentence in order to predict the next word. The MEMM algorithm is trained on a corpus of text that has been annotated with named entities. This annotated data is used to build a model that can then be used to predict named entities in a new text. [4]

One of the advantages is that it can be easily adapted to different languages and domains. Additionally, this algorithm is relatively fast and can be run on large corpora of text. It simply

scans through a text document and looks for strings of capitalized words. These strings are then classified as named entities. [4]

1.2.2 BERT for NER

BERT, or Bidirectional Encoder Representations from Transformers, is a pre-trained language model that can be fine-tuned for a variety of downstream tasks such as classification, Question Answering, and NER. [5] BERT can be used for NER by fine-tuning the pre-trained model on a labeled dataset of text that contains named entities.

The Transformer is a neural network architecture that dispenses with recurrence, entirely relying only on the attention mechanism to draw global dependencies between input and output. Since Transformers do not rely on sequential processing, they can process an input sequence of words all at once, allowing for much more parallelization and requiring significantly less time to train compared to Bi-LSTM-based models [6] (which will be described later on in Chapter 5).

1.3 Usage of NER

NER, in general, can have many uses apart from just locating and labeling entities in text. It can improve the speed and relevance of search results and recommendations by summarizing descriptive text, reviews, and discussions. [7] For building a structured database by recognizing entities of the desired type from a collection of texts. For example, we might build a database of medical disease names by scraping the web and recognizing entities of this type. [2] NER is also useful for content classification by surfacing content more easily and gaining insights into trends by identifying the subjects and themes of blog posts and news articles. [7] Basically, anywhere where it is useful to extract and/or to categorize some data that is in a text form, NER can be useful.

1.3.1 Usage of NER on Czech Poetry

Almost all the cases mentioned above can be applied to our dataset of Czech poetry. Whether it is to classify poems based on which entities occur in them or to help create a potential search that would search for poems based on a query. Or any other use case that would benefit from the added context of knowing what entities of which category each poem has.

1.4 Entity labeling

There is also a need to effectively represent entities in the text so we know which words are entities and which are not. NER can also be viewed as a sequential prediction problem in which we aim to assign the correct label for each token. There are more ways in which we can encode the information about the token into the label. The most popular ones are called BIO and BILOU. [8] Since the model that was trained for the purpose of this thesis is made for the use of BIO tagging, it will be the type of tagging that we will use here. Even though there is a case to be made for using BILOU since it was found to outperform BIO. [9]

1.4.1 BIO tagging

BIO stands for Beginning, Inside, and Outside (of a text segment). In a system that recognizes entity boundaries only, only three labels are used: B, I, and O. [8] In Figure 1.2 is an example of a BIO scheme with entity categories.

```
Minjun B-Person
is      O
from   O
South  B-Location
Korea  I-Location
.      O
```

■ **Figure 1.2** BIO tagging [8]

Corpus of Czech Verse

This chapter describes the Corpus of Czech Verse (CCV) which is a corpus this thesis will be working with.

“The Corpus of Czech Verse (CCV) is a lemmatized, phonetically, morphologically, metrically, and strophically annotated corpus of Czech poetry of the 19th century and the beginning of the 20th century. Each lexical unit in the corpus is provided with information concerning its basic entry form (lemma), phonetic transcription, and grammar categories; each line is provided with the indication of its metre (iamb, trochee,...), length (n-meter), ending type (masculine, feminine,...) and the metrical formula.” [1]

2.1 Corpus Scheme

CCV is divided into multiple books in JSON file format, where each book contains a number of poems. Each poem then includes a number of lines (verses) and is made up of multiple words. Books, poems, lines, and words all contain metadata about themselves. Like the title, year, author, etc., for books. Or text, words, metre, rhyme, etc., for lines.

2.1.1 Poem Schema

Each poem contains a lot of metadata, mainly about the book in which the poem was published. Information about the author of the book, but also about the author of the poem itself, plus information like the number of pages, publisher, etc. For NER purposes, the most valuable information is stored in the body tag. This tag contains a list of stanzas, and each stanza contains a list of lines. The poem schema can be seen in Figure 2.1.

2.1.2 Line Schema

Every line contains the text of said line in UTF-8 encoding, the type of rhyme used in this line, a dictionary with information about the punctuation in the given line, and a list of words that are used in this line. Each word is saved as a dictionary with keys like phoebe [10] or xsampa [11]

(both used for phonetic expression of the word). But also token (word in the format as it appears in the text), token_lc (same as a token but lowercased), lemma, and morph (morphological tag using the Prague positional system).

2.2 Prague positional system

The CCV is using Prague positional system for morphological tagging. We were, unfortunately, unable to find any text that would have described or even mentioned Prague positional system. All we know is that morphological tags used in the dataset are 16 characters long. Most sources for the description of Czech morphology we found use 15 characters or less. The good thing is that the morphological categories we will be interested in are Part of Speech and Case. All of the sources we looked at put these categories in positions 1 and 5. Also from looking at our dataset, these categories seem to be in positions 1 and 5. There may be some differences between the tagging used by our dataset and the ones used in Figure 2.3 but these differences are mostly in categories in further positions. In Figure 2.2 we can see an example of a morphological tag used in the CCV and in Figure 2.3 we can see how potentially (with subtle differences) the tag structure could look like.

“Every tag is represented as a string of 15 symbols. Each position in the string corresponds to one morphological category according to a more or less traditional system of formal morphology.” [13] A value in each category is represented as a single symbol, mostly an uppercase letter of the English alphabet (for example, *P* for plural), sometimes also another symbol (*f* for an infinitive) or a number (*4* for fourth case). [13]

```

{
  'book_id': ID OF THE BOOK,
  'poem_ids': ID OF THE POEM,
  # Metadata on author or the editor of the entire book
  'b_author': {
    ... SOME METADATA ABOUT THE AUTHOR ...
  },
  # Metadata on author of the poem
  'p_author': {
    ... SOME METADATA ABOUT THE AUTHOR ...
  },
  'biblio': {
    ... SOME METADATA ABOUT THE BOOK ...
  },
  # The poem itself encoded as a list of lists (stanzas x lines)
  'body' : [ [
    {
      'text': TEXT OF THE LINE,
      'stress': BITSTRING ENCODING STRESSED (1) and UNSTRESSED (0) SYLLABLES,
      'rhyme': RHYME INDEX (the lines that rhyme all share the same value here),
      # List of meters assigned to the line
      # (in rare cases of ambiguous line, different meters may be assigned to it)
      'meter': [{
        'type': J=iamb / T=trochee / D = dactyl / A = amphibrach / X = dactylotrochee /
              Y = dactylotrochee with anacrusis / hexameter / pentameter / N = not recognized,
        'clause': TYPE OF LINE ENDING (f=feminine, m=masculine, a=acatalectic),
        'foot': NUMBER OF FOOT,
        'pattern': PATTERN OF STRONG(S) AND WEAK(W) POSITIONS,
      }, ...],
      # List of words and their metadata
      'words': [{
        'token': WORD AS APPEARS IN THE TEXT,
        'token_lc': LOWERCASED token,
        'lemma': LEMMA,
        'morph': MORPHOLOGICAL TAG (Prague positional system),
        'xsampa': PHONETIC TRANSCRIPTION IN XSAMPA,
        'phoebe': SIMPLIFIED PHONETIC TRANSCRIPTION (CCV INTERNAL FORMAT),

        }, ...],
      # Dict holding punctuation marks
      # Punctuation marks are stored under the key which corresponds
      # to the index of a word which the punctuation precedes
      # (for line-final punctuation key == len(words)
      'punct': {},
    },
    ...
  ], ... ]
}

```

■ Figure 2.1 Poem Schema [12]

Bůh (NNMS1-----A-----)

■ **Figure 2.2** Example of morphological tag from CCV dataset (16 positions)

#	Category Name	Description in English	Description in Czech
1	POS	Part of Speech	Slovní druh
2	SUBPOS	Detailed Part of Speech	Slovní poddruh
3	GENDER	Gender	Rod
4	NUMBER	Number	Číslo
5	CASE	Case	Pád
6	POSSGENDER	Possessor's Gender	Rod vlastníka
7	POSSNUMBER	Possessor's Number	Číslo vlastníka
8	PERSON	Person	Osoba
9	TENSE	Tense	Čas
10	GRADE	Degree of Comparison	Stupeň
11	NEGATION	Negation (by prefix)	Negace
12	VOICE	Voice	Slovesný rod
13	ASPECT	Aspect	Vid
14	AGGREGATE	Aggregate	Agregát
15	VAR	Variant	Varianta

■ **Figure 2.3** Tag Structure as found in [13] (15 positions)

Entity selection

This chapter explains how entities are selected using our rule-based approach.

Since NER usually needs a supervised machine learning model, we need to create a training dataset on which a model can be trained later on. We could theoretically go through a large enough number of poems and label each and every word with its entity category by hand. This is, however, not really feasible given the time constraint and the amount of manpower we have at our disposal. Therefore we need to create a set of rules which define whether or not a given word is an entity, and in case it is, we categorize this entity using another set of rules. Then we select a set of stanzas that have all of its entities reliably categorized and use this set as our training data on which we will train a model later on. In other words, we are solving NER as something in between a supervised and an unsupervised task.

But before we can start categorizing our entities, we need to find them in the text. To do this, we need to create a set of rules and use those rules to select our entities.

3.1 Metrics used for evaluation of the rule-based approach

As said before, in the CCV, it is not marked which words are entities and which are not. Therefore it is pretty difficult to create accurate metrics that would measure how well our set of rules selects entities. So whenever we created a new set of rules, we simply tested it by trial and error. We ran the set of rules on the CCV dataset and looked at the results. Looked into the errors the program made trying to find some patterns and created a new set of rules that sorted the problems we found. We did this until we found the rules that selected words/sequences of words that were almost certainly entities.

It is important to note here that we only reviewed which words were selected as entities. We were not looking at the whole text and searching for entities that were left behind (that would be extremely time-consuming). Because our main goal was to create as clean a set of entities as possible (that can be used for model training later), therefore we were trying to minimize the number of false positives even at the expense of increasing the number of false negatives. But we tried to have the set of entities as big and diverse as possible as well (the bigger and more diverse the training dataset, the better the model usually).

3.2 Capital words

Since the entities that we are focusing on are entities of either Person or Place (or some variations of these), we focused a lot on the fact if a word starts with a capital letter or not. This is because the rules of the Czech language dictate that people's and place's names have to begin with a capital letter, at least in the first word. We have 1,271,237 words whose tokens start with a capital letter in our dataset which makes quite a decent amount of words that could potentially be entities.

3.3 Single-word entities

At first, let us focus on entities that have a length of only one word (like *Václav*, *Bůh*, *Třebíč*, etc.) and not entities like *Jan Ámos Komenský*, *Bílá Hora*, etc.

3.3.1 Capital lemmas x Capital tokens

We tried to select single-word entities by selecting words with capital first letters in their lemma format. The lemma formats in the CCV dataset were created using the MorphoDiTa program [14]. We do not know how MorphoDiTa's model creates lemmas from tokens and decides which ones to capitalize and which ones to leave lower-cased. But the words with capital first letters in their lemma format seem to look more often entity-like.

Then we also tried to select words with a capital first letter in their token format and that are not at the beginning of a sentence. Word is marked as being at the beginning of a sentence when it is either the first word of a poem's line or when it is right after punctuation that marks the end of a sentence (*!*, *!*, *?*, *:*, *..*, *,*, *„*, *”*). The number of potential entities if we use capital tokens, is 209,654. If we use lemmas, it is 210,755 words. So there is not much difference here in the number of potential entities.

However, there is quite a big difference in which words get chosen. The set created by the lemma rule has 70,938 words that are not chosen using the token rule. On the other hand, the set created by the token rule has 115,232 words that are not chosen by the lemma rule. After inspecting which words were chosen using capital lemmas and which using capital tokens, we found out that neither of these rules works perfectly. Maybe using capital lemmas seems the more accurate solution, but it's also not the ideal way. On the other hand, the problem with using capital tokens is that we are only picking words that are not at the beginning of a sentence (if we also picked words that are at the beginning of a sentence, we would have too many words that are not entities and are just the first words in a sentence). So later on, when we want to use these words as our training data, our model could potentially pick up on this pattern and miss a lot of entities that are at the beginning of a sentence.

Later on (as seen in Subsection 3.3.4), we decided to implement both rules. So words that are potential entities have to have a capital first letter in their lemma and token format.

3.3.2 Using Morphology

If we would select words that are capitalized in their lemma and token format, we can still see a lot of words that are clearly not an entity (*II.*, *Hlucho*, *Osud*, *Hurrá*, etc.). Most of these words are morphologically something other than nouns. Therefore we tried to select only those words

that have capital lemma but also are nouns. This seemed to work quite well, but the problem is that we may miss a lot of potential entities this way. Because in the Czech language, we can also find entities that are not nouns, a good example would be surnames since a lot of them are actually adjectives (*Veselá*, *Smutný*, *Černý*, etc.). Another problem is that some words do not have an assigned morphology since their POS (part of speech) is marked as being unknown. If we look closely at morphologically unknown words, we can see some of them being entities (*Vatikanu*, *Douamont*, *Bastilla*, etc.).

3.3.3 Forbidden Lemmas

After closer inspection of our potential entities, we also found out that there are some words that do have the capital first letter in their lemma format, and yet they are clearly not entities. Some of these words were quite commonly recurring in our set of potential entities. Therefore, we decided to create a list of forbidden lemmas. If we encounter a word that has its lemma format in this list, we do not count this word as an entity. These words are [*"Ty's"*, *"Toba"*, *"Tvojm"*, *"Ont"*, *"Mni"*, *"Huja"*, *"Cos"*, *"Jakoby"*, *"Celá"*, *"Ton"*, *"Jež"*, *"Jaj"*, *"Tyt"*, *"Mni"*, *"Jakby"*, *"Ant"*, *"Bouř"*].

3.3.4 Final version

Word is marked as being an entity if it complies with the following rules:

- Its lemma format starts with a capital letter.
- Its token format starts with a capital letter.
- If the morphology of the word is unknown, the word cannot be at the beginning of a sentence or line
- If its lemma is not in the list of forbidden lemmas.
- If the length of its lemma is longer than 3. There are barely any entity names in the Czech language that would be smaller than 3, and there were quite a few lemmas that had a capital first letter, and their length was smaller than 3 (usually Roman numbers, some connectors, and adverbs).

3.4 Multi-word entities

As said above, not all entities contain just one word. There are quite a few entities that span across two or more words. These could be names (*Josef Slaný*, *Petra Dlouhá*) or places (*Bílá Hora*, *Nové Město na Moravě*).

3.4.1 Czech grammar on multi-word entities

Since the entities that we are interested in often contain at least one word with a capital first letter, we focused on how capital letters are used in multi-word names in the Czech language. Czech multi-word proper names are written with a capital letter in the first word in principle (*Bílá hora*, *Krušné hory*, *Nová řeka*, *Národní divadlo*). Provided that a proper name is a component of another multi-word proper name, it is written with a capital letter as well (*Tichá Orlice*,

Studená Vltava, Moravskoslezské Beskydy). Further supplementary rules hold for geographical names. [15]

In multi-word proper names of residential objects (in officially used denotations of towns, their parts and districts, villages and hamlets), all words except prepositions are written with capital initials (*Karlovy Vary, Klenčí pod Čerchovem, Nové Město pod Smrkem*, etc.). [15]

There are even more supplementary rules regarding geographical names, which can be found in (Boháč, 2007) [15].

For a person's name, the Czech language follows the same convention as anywhere else in the world. Which is that all parts of someone's name begin with a capital letter (*Jan Hus, Jan Ámos Komenský*, etc.).

3.4.2 Position of words in entity

If one entity covers multiple words, these words usually sit right beside each other, and their first letters are capitalized. There are also exceptions to this rule, for example, *Nové Město na Moravě* or *Jiří z Poděbrad* and many more. We, however, decided not to focus on this specific subset of entities (we only focused on multi-word entities whose words are right beside each other and have capital first letters). This is mainly because creating a simple and effective set of rules to include entities like *Jiří z Poděbrad* but also at the same time not include just some random multi-word phrases proved to be quite difficult. Another reason is that most of these entities are included nonetheless, just not as one entity but, for example, as two or more (*Jiří z Poděbrad* will be separated into entity *Jiří* (Person) and *Poděbrad* (Place) instead of one Person entity).

3.5 Final version of rule-based approach

After trying out different rules on how to select all entities (single-word and multi-word as well). We have arrived at a final approach that is described in this Section.

The rules can be described in 9 steps. For each book in the CCV dataset, all of these steps are applied one by one to extract entities from the dataset. After a description of each step, an example of words that are currently selected as entities is shown (by currently meaning after that step).

1. Firstly, we create a set of word sequences that could potentially be entities, and let us call it *ents_mw* (detailed description of rules for creating this set is in Subsection 3.5.1). This set consists of sequences of words (even of a length one) with capital first letters in their token format. Morphologically allowed parts of speech are only nouns, adjectives, numbers, and morphologically unknown words. However, the sequence of words can only start with a noun, an adjective, or a morphologically unknown word.

► Example 3.1.

Example description:

Example set = {

"Entity in lemma form" ("Morphological tag") ["Line from the book in which this entity occurs for the first time", "Line from the book in which this entity occurs for the second time"],

"Another entity in lemma form", ("another morphological tag") ["use case"],

...
}

► **Example 3.2.**

```
ents_mw = {
Karel (NNFS1—A—) dvanáctý (CrMS2—) ["Karla Dvanáctého, krále Švédů"],
Bůh (NNMS2—A—) otec (NNMS2—A—) ["„Jménem Boha Otce, Boha Syna”],
Petr (NNM-2—) ["Bez Petra není zábavy"],
Konstanci (NNFS6—A—) Husa (NNMS4—A—) — ['co v Konstanci Husa upálili.'],
Ctnost (NNFS1—A—) Plutarchovská (X24—) — ['Ctnost Plutarchovská ve tmách
dneška plane,']
}
```

2. Secondly, we remove from *ents_mw* all entities that contain at least one word that is morphologically unknown. We have to remove the whole potential entity, not just the word that is morphologically unknown, because then we could have been left with a word that by itself is not an entity. The number of these potential entities containing morphologically unknown word(s) can be later on seen in Table 3.1.

► **Example 3.3.**

```
ents_mw = {
Karel (NNFS1—A—) dvanáctý (CrMS2—) ["Karla Dvanáctého, krále Švédů"],
Bůh (NNMS2—A—) otec (NNMS2—A—) ["„Jménem Boha Otce, Boha Syna”],
Petr (NNM-2—) ["Bez Petra není zábavy"],
Konstanci (NNFS6—A—) Husa (NNMS4—A—) ['co v Konstanci Husa upálili.']
}
```

3. In *ents_mw* all words in one entity have to be in the same case (morphological case). This applies only to words that are a noun or an adjective (not numbers). This is done to prevent word sequences like *Kostnici Husa* in the sentence *co v Konstanci Husa upálili.* from being marked as one entity. In the Czech language, if we talk about something as one whole entity, we tend to put all of the words of this whole entity into the same case. An example could be the name *Josef Slaný* (the cases for this name would be: 1. *Josef Slaný*, 2. *Josefa Slaného*, 3. *Josefu Slanému*, etc.), as we can see if we decline the entity as one whole entity, all of the words of this entity are put in the same case. If we find entities whose words are not in the same case, we split them so we are only left with entities that have all of their words in the same case.

► **Example 3.4.**

```
ents_mw = {
Karel (NNFS1—A—) dvanáctý (CrMS2—) ["Karla Dvanáctého, krále Švédů"],
Bůh (NNMS2—A—) otec (NNMS2—A—) ["„Jménem Boha Otce, Boha Syna”],
Petr (NNM-2—) ["Bez Petra není zábavy"],
Konstanci (NNFS6—A—) ['co v Konstanci Husa upálili.'],
Husa (NNMS4—A—) ['co v Konstanci Husa upálili.']
}
```

4. Remove from *ents_mw* entities that only contain one word.

► **Example 3.5.**

```
ents_mw = {
Karel (NNFS1—A—) dvanáctý (CrMS2—) ["Karla Dvanáctého, krále Švédů"],
Bůh (NNMS2—A—) otec (NNMS2—A—) ["„Jménem Boha Otce, Boha Syna”]
}
```

5. Get single-word entities using the rules from Subsection 3.3.4. Let us call the set of single-word entities *ents_sw*.

► **Example 3.6.**

```
ents_sw = {
Bůh (NNM-1————) [”„Jménem Boha Otce, Boha Syna”, ”Bůh se o to postará”, ”Bez
požehnání Boha neudělá krok”],
Petr (NNM-2————) [”Bez Petra není zábavy”],
Karel (NNFS1—A—) [”Karla Dvanáctého, krále Švédů”],
Konstanci (NNFS6—A—) [’co v Konstanci Husa upálili.’],
Husa (NNMS4—A—) [’co v Konstanci Husa upálili.’]
}
ents_mw = {
Karel (NNFS1—A—) dvanáctý (CrMS2————) [”Karla Dvanáctého, krále Švédů”],
Bůh (NNMS2—A—) otec (NNMS2—A—) [”„Jménem Boha Otce, Boha Syna”]
}
```

6. Add a unique ID to every word in a book. Do this in all books (therefore IDs are only unique in the context of one book). This is necessary for the following step.

► **Example 3.7.**

```
ents_sw = {
Bůh (NNM-1————) [”„Jménem(ID: 1) Boha(ID: 2) Otce(ID: 3), Boha(ID: 4) Syna(ID: 5)”,
”Bůh(ID: 45) se(ID: 46) o to postará”, ”Bez požehnání Boha(ID: 72) neudělá krok”],
Petr (NNM-2————) [”Bez Petra(ID: 22) není zábavy”],
Karel (NNFS1—A—) [”Karla(ID: 37) Dvanáctého, krále Švédů”],
Konstanci (NNFS6—A—) [’co v Konstanci(ID: 52) Husa(ID: 53) upálili.’],
Husa (NNMS4—A—) [’co v Konstanci(ID: 52) Husa(ID: 53) upálili.’]
}
ents_mw = {
Karel (NNFS1—A—) dvanáctý (CrMS2————) [”Karla(ID: 37) Dvanáctého, krále Švédů”],
Bůh (NNMS2—A—) otec (NNMS2—A—) [”„Jménem Boha(ID: 2) Otce,
Boha(ID: 4) Syna”]
}
```

7. If an entity from *ents_sw* is part of an entity in *ents_mw* remove it from *ents_sw*. To know if a single-word entity is part of a multi-word entity, we check their word IDs. If the word ID of a single-word entity matches with at least one of the words ID of a multi-word entity, we remove this entity from *ents_sw*. It is important to note that if a single-word entity occurs on multiple positions in the text, the IDs must match in all of these occurrences in order for the entity to be removed from *ents_sw*.

► **Example 3.8.**

```
ents_sw = {
Bůh (NNM-1————) [”„Jménem Boha(ID: 2) Otce, Boha(ID: 4) Syna”, ”Bůh(ID: 45) se o
to postará”, ”Bez požehnání Boha(ID: 72) neudělá krok”],
Petr (NNM-2————) [”Bez Petra(ID: 22) není zábavy”],
Konstanci (NNFS6—A—) [’co v Konstanci(ID: 52) Husa(ID: 53) upálili.’],
Husa (NNMS4—A—) [’co v Konstanci(ID: 52) Husa(ID: 53) upálili.’]
}
```

```
ents_mw = {
  Karel (NNFS1—A—) dvanáctý (CrMS2—) ["Karla(ID: 37) Dvanáctého, krále Švédů"],
  Bůh (NNMS2—A—) otec (NNMS2—A—) [",,Jménem Boha(ID: 2) Otce,
  Boha(ID: 4) Syna"],
}
```

8. If a word in an entity from *ents_mw*, is in *ents_sw*, then remove this entity from *ents_mw*. If a lemma format of a word that is part of an entity in *ents_mw* is the same as a lemma format of any word in *ents_sw*, then remove the entity containing this word from *ents_mw*. This is done so we can be sure that *ents_mw* truly contains only multi-word entities. Without this rule, there would be a possibility that we can have something marked as a multi-word entity, but it's just two single-word entities next to each other. This way, *ents_mw* only has entities containing words that by themselves are not entities in the same book.

► **Example 3.9.**

```
ents_sw = {
  Bůh (NNM-1—) [",,Jménem Boha Otce, Boha Syna", "Bůh se o to postará", "Bez
  požehnání Boha neudělá krok"],
  Petr (NNM-2—) ["Bez Petra není zábavy"],
  Konstanci (NNFS6—A—) ['co v Konstanci Husa upálili.'],
  Husa (NNMS4—A—) ['co v Konstanci Husa upálili.']
}
ents_mw = {
  Karel (NNFS1—A—) dvanáctý (CrMS2—) ["Karla Dvanáctého, krále Švédů"]
}
```

9. Join *ents_sw* and *ents_mw* into one set.

3.5.1 Selecting potential entities

This subsection explains in detail how the first step in the final version of entity selection works (the previous Section 3.5). The goal is to select all word sequences (even of a length one) that could potentially be entities. To do that, the function seen in Code listing 3.1 is used. It works by iterating over each line of a poem in a book. For each line, it then iterates over each word. Conditions are created in a way that each word sequence (potential entity) has to meet these rules:

- The first word of the sequence has to meet these conditions (all of the conditions below have to be met):
 - Its token form starts with a capital first letter.
 - The word is either a noun, an adjective, or a morphologically unknown word.
 - If the word is at the beginning of a line or a sentence, then its lemma form also has to have a capital first letter.

- Every other following word of the sequence has to meet these rules in order to be a part of the same potential entity as the word before:
 - Have in its token form a capital first letter.
 - Be either a noun, an adjective, a preposition, a number, or a morphologically unknown word.
 - Punctuations cannot be a part of an entity. Therefore, if any punctuation occurs, the sequence of words that make up one potential entity is stopped.

3.6 Result of entity selection

After applying the rules from Section 3.5, We have found 163,100 potential entities. But in these 163,100 entities, we only found 24,655 unique entities (unique in the sense that we counted every unique lemma form of an entity only once), which is about 15%. This means that there are a lot of entities that repeat themselves in the text, which could be a good thing for us because it means we do not have to categorize that many of them.

If we look at Table 3.1 we can see that most of our selected entities are single-word entities, with only a few entities being multi-word. We can also see how many other potential entities we might have missed by removing morphologically unknown words (rule from step number 2 in Section 3.5).

■ **Table 3.1** Entity selection overview

Set of words	Count
All entities (non-unique)	163,100
Unique entities	24,655
Single-word entities (non-unique)	160,469
Multi-word entities (non-unique)	2,631
Potential entities containing a morphologically unknown word(s) (non-unique)	2,572
Words that are morphologically unknown (non-unique)	358,575
Capital words (non-unique)	1,271,237

■ **Code listing 3.1** Selecting potential entities

```
def ents_select_possible(book: Book) -> list:
    possible_entities = list()

    for line in book.get_lines():

        # Create empty Entity class (entity not containing any words)
        entity = Entity([])

        # For each word and its position (index) on the line
        for index, word in enumerate(line.words):

            # If the entity is NOT empty and
            # the word is NOT one of the following:
            # noun, adjective or unknown
            if not entity and \
            not allowed_morphs(word, constants.ALLOWED_MORPH_START):
                continue

            # If the word is at the beginning of a line or
            # at the beginning of a new sentence
            # i.e. is right after [".", "!", "?", ":", "...", ",", "]
            # and its lemma form does NOT have a capital first letter
            # then save the entity into all possible entities
            if (index == 0 or \
            end_sentence(punct_on_pos(line, index))) and \
            not word.is_capital_lemma():
                possible_entities.append(entity)
                entity = Entity([])
                continue

            # If the word is NOT one of the following:
            # noun, preposition, adjective, number or unknown
            if not allowed_morphs(word, constants.ALLOWED_MORPH_2):
                possible_entities.append(entity)
                entity = Entity([])
                continue

            # If there is punctuation right in front of this word
            if punct_on_pos(line, index):
                possible_entities.append(entity)
                entity = Entity([])
                continue

            # If the word in its token form does
            # NOT have a capital first letter
            if not word.is_capital_token():
                possible_entities.append(entity)
                entity = Entity([])
                continue

            # If none of the conditions in front was true
            # then add/push the word as a part of the entity
            entity.push(word)

        # If the entity is not empty after the end of a line
        # then save the entity into all possible entities
        if entity:
            possible_entities.append(entity)

    return possible_entities
```


Categorization of Entities

This chapter explains how entities are categorized using our rule-based approach.

After we have managed to select which words are entities, we now have to categorize them. After inspecting the selected entities, we have decided to categorize them into five different categories. Those categories are Real Person, Mystic Person, Person, Place, and Other. The reason for these entity categories is the same as already mentioned in Subsection 1.1.1, and that is that any other entities of similar meaning do not seem to appear in the dataset in large enough quantities to create a category for them.

4.1 Categories description

The category of a Real Person contains historical personas, dead or alive, that have actually lived (for example, *Karel IV.*). Mystic Person includes characters used in mythologies (*Medúsa*) or a well-known fictional character (*Drákula*), or a form of deity (*Bůh*). In the Person category, we find any other person that is not a Real or Mystic Person. Usually, first names like *Jan*, *Petr* or *Pavla* or general surnames like *Novák*, *Blažková* or *Škvarenina* are in this category. Lastly, the category of Other describes the rest of the entities (for example, abstract entities like *Láska* or *Štěstí*).

4.2 Implementation of categorization

Czech Wikipedia and Wikidata help us to categorize the entities. Program in Python is implemented to go through all of the entities we found and search for each and every one on the Czech Wikipedia page. If a Wikipedia page for this entity is found, it is then scraped to find out what type of entity it represents. If the Wikipedia page for this entity is not found, we try to search for it on Wikidata and scrape the answers we get there.

4.2.1 Wikipedia page layout

If we look into Figure 4.1 and 4.2, the highlighted sections with numbers are the parts of the Wikipedia page that are scraped for information so we can identify the correct category for a given entity. In Figure 4.1 oval 1, we can see the geographical location. This location is only shown if the Wikipedia page is about a place. In rectangle number 2, we can see what is called the summary of the Wikipedia page. In this summary, we can find some keywords that help us identify what type of entity the article might talk about. In rectangle number 3, we can find what is called an infobox of a Wikipedia page. This infobox is also used to get some information that might be valuable to us. And lastly, in Figure 4.2, we can see the bottom of the same Wikipedia page. At the bottom of each page, we can find categories to which this page belongs.

■ **Figure 4.1** Top of Wikipedia page

The screenshot shows the top of the Wikipedia page for "Karlovy Vary".

- 1:** A red oval highlights the top right navigation area, including the language selector "73 jazyků" and links for "Číst", "Editovat", "Editovat zdroj", and "Zob. hist.", along with a location map showing coordinates 50°13'50" s. š., 12°52'21" v. d.
- 2:** A red rectangle highlights the summary paragraph: "Karlovy Vary (německy *Karlsbad*) jsou krajské a statutární město v okrese Karlovy Vary v západních Čechách, v Karlovarském kraji, 110 km západně od Prahy na soutoku Ohře a Teplé. Žije zde přibližně 46 tisíc^[1] obyvatel. Je zde rozvinut mj. sklářský a potravinářský průmysl. Jedná se o nejnavštěvovanější české lázeňské město.^[4] Od roku 2021 je na seznamu Světového dědictví UNESCO v rámci položky Slavná lázeňská města Evropy. Město je členem sdružení Regionální sdružení obcí a měst Euregio Egrensis a Vodohospodářské sdružení obcí západních Čech.
- 3:** A red rectangle highlights the infobox content, including the "Historie" section, a historical map from 1650, and the "Statutární město Karlovy Vary" section with the coat of arms and flag.

4.2.2 Categorizing entities using its Wikipedia page

The categorization of entities using their Wikipedia pages is going to be described as a step-by-step process. It is important to note here that once the entity has been categorized, the categorization process ends. For example, if an entity called *Nepomuk* gets categorized as an entity of Place in step 2, the categorization process will not continue to step 3 to check whether or not entity *Nepomuk* cannot also be an entity of a Real Person.

1. Firstly, we try to find a Czech Wikipedia page about a given entity by searching for this entity in the Wikipedia search. The term we search for is the entity name in a lemma format. This seems to be more effective at finding Wikipedia pages than searching in token format.

■ **Figure 4.2** Bottom of Wikipedia page




■ **Figure 4.3** Search in token format



■ **Figure 4.4** Search in lemma format

This way of searching is quite effective for single-word entities. But as we can see in the example in Figure 4.3 and 4.4, neither of these approaches is ideal for multi-word entities (since the actual entity name that we are trying to search for is called *Bílá Hora*). For searching multi-word entities in the Czech language, the ideal approach would probably be to transform the token format found in the text into the first (nominative) case and search in this format (i.e. switching from *Bílou Horou* to *Bílá Hora*). Unfortunately, our CCV does not contain the first (nominative) case format for each word. We failed to find any API that would convert words into their nominative forms, and creating an API that would reliably work was not feasible.

Therefore, if the Wikipedia page is found using the entities lemma format, we carry on trying to categorize it (we move to step 2). If the Wikipedia page is not found, we try to categorize the entity using Wikidata (Subsection 4.2.4).




2. Second step is to find out if a given Wikipedia page is about the entity of Place. This is done by trying to find the geolocation tag (number 1 in Figure 4.1) on this Wikipedia page. If the geolocation tag is found, we classify the entity as an entity of Place.
3. If a given entity does not have a geolocation tag, we try to find out if it is a Real Person. This is done by searching for either the word *Narození* or *Datum narození* in the infobox (number 3 in Figure 4.1 of the Wikipedia page. If either of these phrases is found, this entity is marked as an entity of a Real Person.
4. Fourth step is checking whether or not a given entity is a Mystic Person. This is done by searching for the word *postava* in the summary of the Wikipedia page (rectangle number 2 in Figure 4.1). The reason is that Mystic Person represents not only entities of diety but also well-known fictional characters (*Drákula*, for instance). And looking at Wikipedia pages of some well-known fictional characters (from mythological stories or fairytales), the word *postava* seemed to appear in their summary quite often. So, if the word *postava* is found in the summary, the entity is marked as Mystic Person.
5. Fifth step is to look through the categories into which this Wikipedia page belongs. By Wikipedia's own definition, "Categories are used in Wikipedia to link articles under a common topic and are found at the bottom of the article page" [16]. We can see where we can find

Wikipedia categories in Figure 4.2. Articles are categorized by the authors or the editors of the Wikipedia articles themselves, so it is not an automated process. [17] Therefore, we had to find commonly occurring categories under Wikipedia pages about entities of Person, Place, etc. These commonly occurring Wikipedia categories are saved as keywords which the categorization program looks for when it is presented with a Wikipedia page.

```
PLACE_CATEGORY = ["místopis", "míst", "stát", "země", "území", "geograf", "vojvodství", "kraj", "říše",
                 "územ", "ostrov", "okres", "středisk"]
PERSON_CATEGORY = ["jmén", "národ", "lid", "král", "postav", "rod"]
MYSTIC_PERSON_CATEGORY = ["boh", "bůh", "bohyně", "mytologi"]
```

■ **Figure 4.5** Keywords used for categorization with Wikipedia page categories

Externí odkazy

-  [Obrázky, zvuky či videa k tématu **Petr** na Wikimedia Commons](#)
-  [Slovníkové heslo **Petr** ve Wikislovníku](#)
-  [Encyklopedické heslo **Petr** v *Ottově slovníku naučném* ve Wikizdrojích](#)

Kategorie: [Mužská jména](#) | [Papežská jména](#) | [Mužská jména řeckého původu](#)

■ **Figure 4.6** Wikipedia category example

We can see all of the keywords listed with their corresponding entity category in Figure 4.5. If any of these keywords are found in the categories of a Wikipedia page, the entity is categorized accordingly. As an example, let us look at Figure 4.6, where we can see the categories for a Wikipedia page about entity *Petr*. Since the keyword *jmén* is a part of the Wikipedia category *Mužská jména*, we can identify the entity *Petr* as an entity of a Person.

- If none of these steps are able to categorize the entity, we try to categorize the entity using Wikidata (Subsection 4.2.3 and 4.2.4).

4.2.3 Wikidata layout

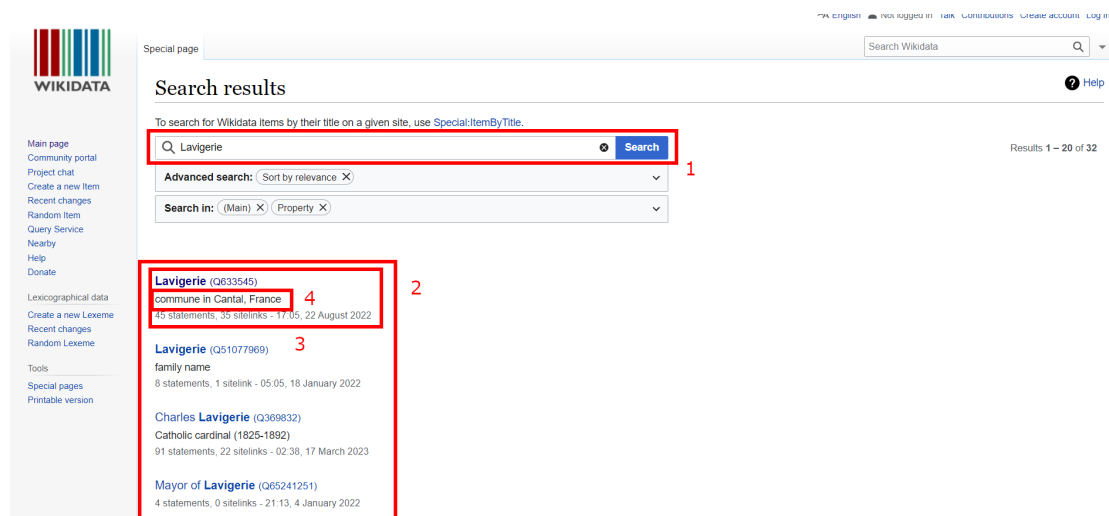
To describe how to categorize entities by scraping Wikidata, we first need to describe the Wikidata page itself.

If we look at Figure 4.7, we can see the Wikidata page layout. In red rectangle number 1, we can see the search box of the Wikidata page. Into this search box, we enter the entity we are trying to categorize. In rectangle number 2, we can see the search results we get. Rectangle Number 3 highlights one individual search result. And lastly, rectangle number 4 highlights the search result description. Information in the search description is used to categorize the entity itself.

4.2.4 Selecting entities using Wikidata

We scrape the description of all search results that the Wikidata search gives us and search for certain keywords to categorize a given entity. We check the search result descriptions (number 4 in Figure 4.7) one by one to try to categorize the entity using the top search results first.

Again, the keywords were selected the same way as keywords for Wikipedia page categories. That means we simply looked at the search results of many entities and then added/removed keywords by trial and error.



■ **Figure 4.7** Wikidata page layout



■ **Figure 4.8** Keywords used for categorization with Wikidata

As an example of how categorization with Wikidata works, we can use Figure 4.7. The entity *Lavigerie* was not successfully categorized using a Wikipedia page (its page was not found/ didn't exist). Therefore, we decided to search for the term *Lavigerie* in Wikidata. After searching for this term, we get the page as in Figure 4.7. At first, we try to categorize *Lavigerie* by the first search result description. This description says: “commune in Cantal, France”. We now try to find our keywords (seen in Figure 4.8) in this description. Since the word *commune* is a keyword representing an entity of a Place, we categorize the entity *Lavigerie* as a Place. If we weren't able to categorize the entity by the first result description, we would move on to the next one until we reached the last search result description.

If even the last search result description did not help us categorize the entity, we would mark the entity as Not Exists.

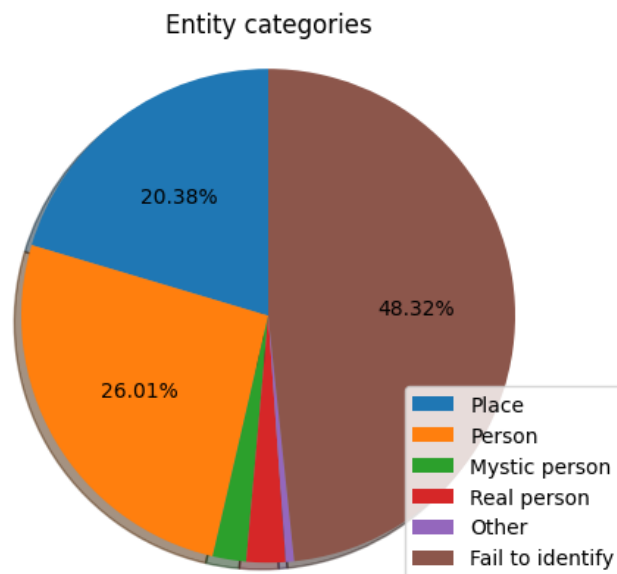
4.2.5 Categorizing entities of category Other

An entity is categorized as an entity of the category Other if we successfully find a Wikipedia page for this entity. But, on this Wikipedia page and also in Wikidata search results, we cannot find anything that would identify the entity as an entity of Place, Person, Mystic Person, or a Real Person.

Now, to distinguish the entity of Other from Nont Exists. For the entity to be categorized as Not Exists, a Wikipedia page for this entity cannot be found. Also, it is not possible for the entity to be categorized by Wikidata search. Whereas for the entity category Other, the entity is also not categorizable using the Wikidata search, but its Wikipedia page exists.

4.3 Result of categorization

We managed to categorize slightly above 50% of all the unique entities selected in Chapter 3, that is, 12,748 out of 24,665 unique entities. In Figure 4.9, we can see how much percent of all the unique entities each entity category covers and also the percentage of entities that we failed to identify (Not Exists) using the Wikipedia and Wikidata approach. Of the entities we managed to categorize, the most prevalent one is the entity of Person, followed closely by the entity of Place. On the other hand, the entities of Real Person (630 out of 24,665), Mystic Person (540 out of 24,665), or Other (137 out of 24,665) are not present in the data that much.



■ **Figure 4.9** Ratios of categorized entities

Theoretical description of our model

Theory of how BiLSTM-CRF neural network works.

In previous Chapters 3 and 4, we selected and categorized entities using a rule-based approach. However, as seen in Figure 4.9, we didn't manage to categorize all of the entities by using this approach. Therefore, we use the entities that we managed to categorize as training data for a machine-learning model. The model that is used for solving NER in this thesis is called BiLSTM-CRF neural network. The reason we picked this model is because it has a free and relatively easy-to-use implementation available at GitHub [18] created specifically for NLP tasks. This model also achieves state-of-the-art performance in many NLP tasks, as reviewed by Reimers et al. (2017). [19] This implementation is also modifiable and has the possibility of fine-tuning multiple hyperparameters (with a white paper [20] attached to it about which hyperparameters to tune for NER or other NLP tasks). More about the implementation itself and the hyperparameter fine-tuning in Chapter 6.

Since the BiLSTM part of the BiLSTM-CRF neural network is a type of recurrent neural network, let us start with describing those.

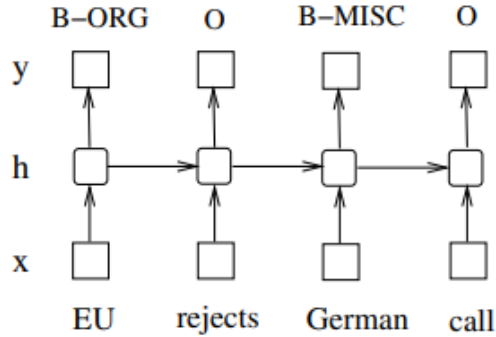
5.1 Recurrent neural network (RNN)

Recurrent Neural Networks (RNNs) are a class of neural networks particularly suited for processing sequential data. Unlike feedforward neural networks, which process inputs in a fixed order and do not have any memory of previous inputs, RNNs can take into account the entire history of a sequence of inputs. [21]

At a high level, an RNN processes a sequence of inputs one at a time, and at each time step, it updates its hidden state based on the current input and the previous hidden state. The hidden state serves as a memory for the network, allowing it to maintain information about the previous inputs that have been processed. [22]

We can see Figure 5.1 below as a graphical demonstration. Here we can see that the input layer (states) is represented by x , the hidden layer by h , and the output layer by y .

We can also see, for example, if we want to determine what entity the word *German* is, we have to consider not only the word itself but also the input from the previous hidden state.



■ **Figure 5.1** RNN example [23]

5.2 BiLSTM-CRF

Before we jump straight into explaining BiLSTM-CRF, let us first explain how LSTM neural networks work. After that, how BiLSTM neural networks work. Thenceforth, what is a CRF, and finally how BiLSTM-CRF combines these neural networks together.

5.2.1 Long Short-Term Memory networks (LSTM)

“Long Short-Term Memory networks are the same as RNNs, except that the hidden layer updates are replaced by purpose-built memory cells. As a result, they may be better at finding and exploiting long-range dependencies in the data.” [23] As we can see in Figure 5.2 below, the memory cell is composed of 4 main components: input gate, output gate, forget gate, and the cell itself.

At time t the following equations are computed in one memory cell [23]:

$$i_t = \sigma(\mathbf{W}_{xi}x_t + \mathbf{W}_{hi}h_{t-1} + \mathbf{W}_{ci}c_{t-1} + b_i) \quad (5.1)$$

$$f_t = \sigma(\mathbf{W}_{xf}x_t + \mathbf{W}_{hf}h_{t-1} + \mathbf{W}_{cf}c_{t-1} + b_f) \quad (5.2)$$

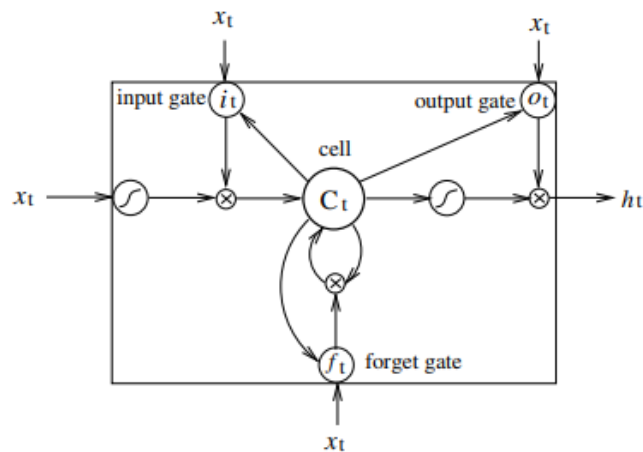
$$c_t = f_t c_{t-1} + i_t \tanh(\mathbf{W}_{xc}x_t + \mathbf{W}_{hc}h_{t-1} + b_c) \quad (5.3)$$

$$o_t = \sigma(\mathbf{W}_{xo}x_t + \mathbf{W}_{ho}h_{t-1} + \mathbf{W}_{co}c_t + b_o) \quad (5.4)$$

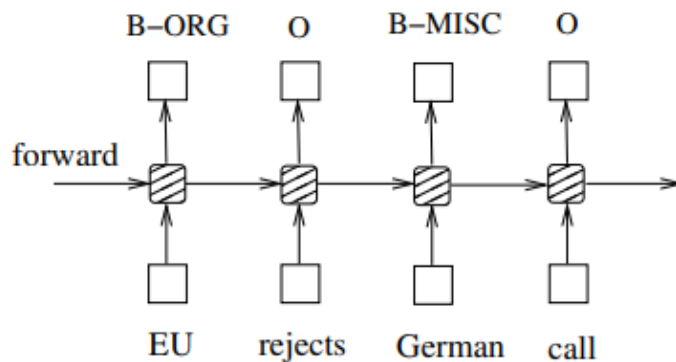
$$h_t = o_t \tanh(c_t). \quad (5.5)$$

where σ is the logistic sigmoid function, and i , f , o , and c are the input gate, forget gate, output gate, and cell vectors, all of which are the same size as the hidden vector h . \mathbf{W} represents the weight matrices. The subscripts below the \mathbf{W} have a meaning as the name suggests. [23] “For example, \mathbf{W}_{hi} is the hidden-input gate matrix, \mathbf{W}_{xo} is the input-output gate matrix etc. The weight matrices from the cell to gate vectors (e.g. \mathbf{W}_{ci}) are diagonal, so element m in each gate vector only receives input from element m of the cell vector.” [23]

In Figure 5.3, we can see a graphical representation of an LSTM network. Each shaded square in the figure represents one memory cell.



■ **Figure 5.2** Memory cell [23]

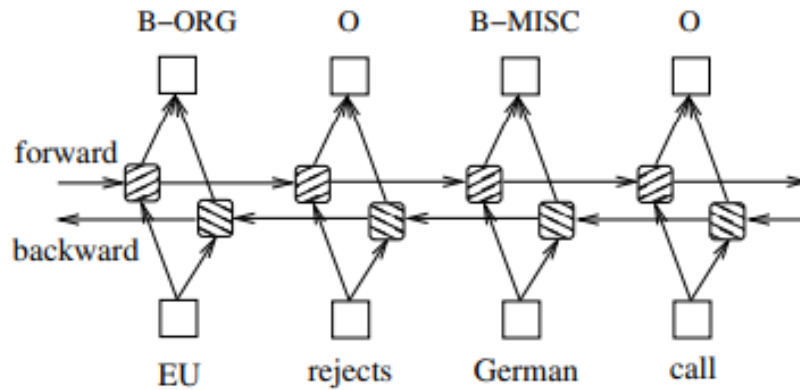


■ **Figure 5.3** LSTM network example [23]

5.2.2 Bidirectional LSTM Network (BiLSTM)

In a sequence tagging task, we have access to both past and future input features for a given time so we can implement a network in a way that makes use of both. Such implementation is, for example, the BiLSTM network. It consists of two LSTM networks. One processes input features using a forward pass (from left to right), and another processes input features by using a backward pass (from right to left). At one point, we need to reset the hidden states (we cannot keep the context of previous/following inputs forever). Since the implementation used in this thesis is based on Huang et al. (2015) [23], we will do forward and backward for whole sentences and only reset the hidden states to 0 at the beginning of each sentence. [23]

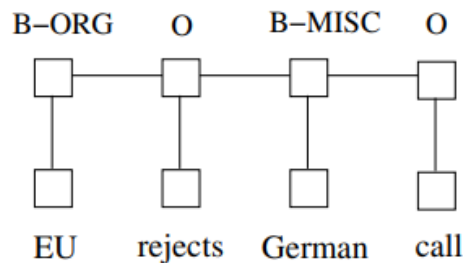
In Figure 5.4, we can see a graphical representation of a BiLSTM network.



■ **Figure 5.4** BiLSTM network example [23]

5.2.3 Conditional Random Fields (CRF)

Instead of focusing on solving NER only one position at a time (evaluating one word at a time, like in BiLSTM), we can try to solve the whole sequence of text at once. That is what CRF focuses on, and it is trying to find optimal tagging for the sequence as one whole. “Note that the inputs and outputs are directly connected, as opposed to LSTM and bidirectional LSTM networks where memory cells/recurrent components are employed.” [23] For more information about CRF, see Lafferty et al. (2001) [24].

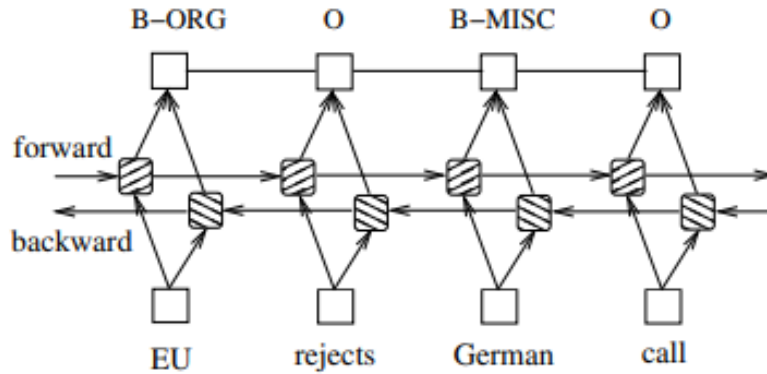


■ **Figure 5.5** CRF network example [23]

5.2.4 BiLSTM-CRF

We combine the BiLSTM network with the CRF network to create BiLSTM-CRF. This network can efficiently use past and future input features via the BiLSTM layer and sentence-level tag information via the CRF layer. [23] In Figure 5.4, we can see an example of BiLSTM-CRF where a CRF layer is represented by lines that connect consecutive output layers and the BiLSTM layer is represented by shaded squares connected by arrows.

If we look at a more detailed graphical description of the BiLSTM-CRF network (Figure 5.7), we can see that firstly each word of a sentence is mapped to a (pre-trained) word embedding. As word embeddings are usually only provided for lower-cased words, we add a capitalization feature



■ **Figure 5.6** BiLSTM-CRF network example [23]

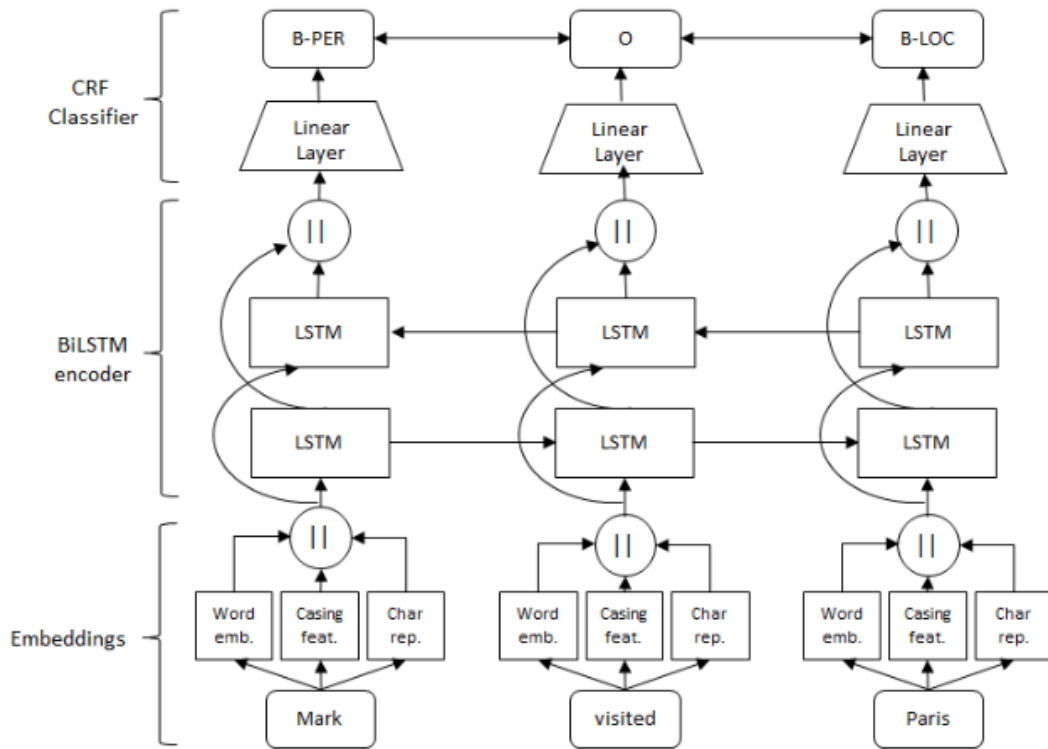
that captures the original casing of the word. There is also an option to derive a fixed-size dense representation based on the characters of the word (using either Convolutional neural network (CNN) or LSTM).

The word embedding, the capitalization feature, and the character-based representation are then concatenated and passed into the BiLSTM encoder (where one LSTM network runs from the beginning of the sentence to the end while the other runs in reverse). The output of both LSTM networks is concatenated and is used as input for a CRF classifier. The input is then mapped with a dense layer and a linear activation function to the number of tags. Then, a linear-chain CRF maximizes the tag probability of the complete sentence. [20]

For an even more detailed description, see Huang et al. (2015) [23] or Reimers et al. (2017) [20] since both of these sources are linked with the implementation used in this thesis.

5.2.5 Theory of training BiLSTM-CRF network

We can see the training procedure illustrated in Figure 5.8. During training, multiple epochs are done. “In each epoch, we divide the whole training data to batches and process one batch at a time. Each batch contains a list of sentences which is determined by the parameter of batch size. In our experiments, we use a batch size of 100 which means to include sentences whose total length is no greater than 100. For each batch, we first run a bidirectional LSTM-CRF model forward pass which includes the forward pass for both the forward state and backward state of LSTM. As a result, we get the output score $f_{\theta}[x]_1^T$ for all tags at all positions. We then run CRF layer forward and backward pass to compute gradients for network output and state transition edges. After that, we can back propagate the errors from the output to the input, which includes the backward pass for both forward and backward states of LSTM. Finally, we update the network parameters which include the state transition matrix $[A]_{i,j} \forall i, j$, and the original bidirectional LSTM parameters θ .” [23]



■ **Figure 5.7** Detailed BiLSTM-CRF network example [20]

```

for each epoch do
  for each batch do
    bidirectional LSTM-CRF model forward pass:
      forward pass for forward state LSTM
      forward pass for backward state LSTM
    CRF layer forward and backward pass
    bidirectional LSTM-CRF model backward pass:
      backward pass for forward state LSTM
      backward pass for backward state LSTM
    update parameters
  end for
end for

```

■ **Figure 5.8** BiLSTM-CRF network: Training procedure [23]

Implementation of our model

This chapter describes this thesis’s specific BiLSTM-CRF implementation and fine-tuning.

The implementation of BiLSTM-CRF that was used in this thesis is publicly available on GitHub [18]. This implementation was created in 2017 and lastly updated in 2018 [18]. It is written in Python and requires Python 3.6 to run (newer versions are incompatible). This older version of Python and older versions of many packages (for example, it uses Keras 2.2.0 and Tensorflow 1.8.0) make it slightly tricky to set up and make it work properly. For example, newer versions of pip (package manager for Python) cannot find some of the package versions mentioned in the *requirements.txt* file. Hence, we had to download some of these packages manually.

6.1 Training data

To create the training data used for training this model, we started by using the categorized entities from the rule-based approach described in Chapters 3 and 4. Thanks to the rule-based approach, we have categorized some of the entities in the CCV dataset (roughly 50% of them).

Now for the training data itself, we used tagged text (every word has a BIO tag with its entity category attached to it) of all stanzas with 100% of the entities found in them categorized. In other words, we used as training data only those stanzas in which all (100%) the entities we found there using the rules from Chapter 3 we were able to categorize using the rules from Chapter 4. So, there should not be a single entity in the stanzas used for training that is not categorized.

In Table 6.1, we can see the number of stanzas with 100% of the entities found in them categorized (let us call them valid stanzas for short). Even though only 50% of the entities were categorized, 94% of the stanzas in the whole CCV dataset can be used for training our model. Even though 94% seems like a big number, as we can also see from Table 6.1, 76% of stanzas do not have any entities in them (and these stanzas are also viewed as valid).

Table 6.1 Set of valid stanzas: Number of stanzas for this set out of the total number of stanzas in the CCV dataset

Number of all valid stanzas	351,434 out of 374,537 that is 93.83%
Number of valid stanzas that do not contain any entity	286,488 out of 374,537 that is 76.49%

6.1.1 Stanzas vs. Poems dataset

At first, however, we didn't use this stanzas dataset and instead, we used all the poems with 100% of the entities found in them categorized (same way of selection, just whole poems instead of whole stanzas). Let us call it the poems dataset. The reasoning behind it was that if we use whole poems instead of just stanzas, we have more context for the model. Later on, however, we found out that the model uses only context within one sentence because the hidden states are reset to 0 at the beginning of each sentence. [23]

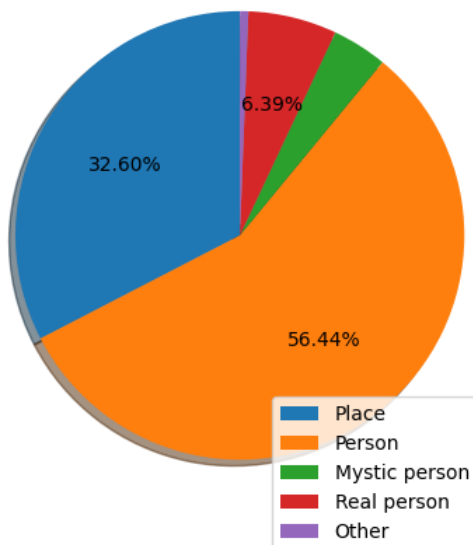
Another reason why we abandoned the poems dataset was that if we were using only whole poems, with all of their entities categorized, we would lose a lot of potential training data. For example, if a poem only has one entity that is not categorized, we cannot use the whole poem in the training dataset.

To see the difference in size between the poems and the stanzas dataset, we can look at Table 6.2 and Figure 6.1. We can see that by using the stanzas dataset, we increase the size of our training dataset a lot (we double the total number of words and triple the total number of entities). But we can also see that the ratios between entity categories stay the same.

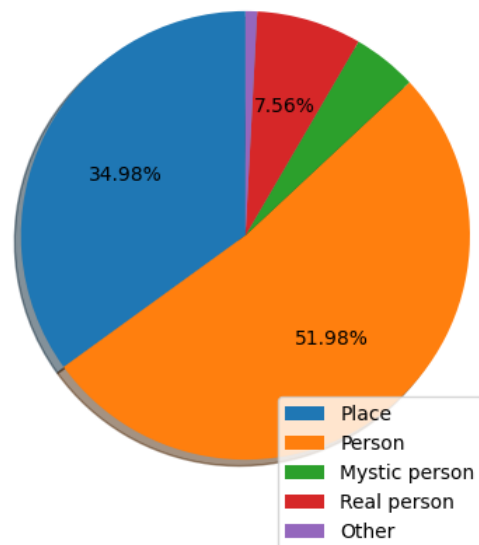
■ **Table 6.2** Poems vs. Stanzas dataset: Number of entities

Number of words in the stanzas dataset	12,402,291
Number of entities in the stanzas dataset	90,941
The ratio of entities to words in the stanzas dataset	0.7333%
Number of words in the poems dataset	6,789,907
Number of entities in the poems dataset	30,205
The ratio of entities to words in the poems dataset	0.4449%

Entity categories in stanzas dataset



Entity categories in poems dataset



■ **Figure 6.1** Poems vs. Stanzas dataset: Entity ratios

6.1.2 Removing stanzas with potentially unmarked entities

Part of the training dataset is not only stanzas that do have some entities but also stanzas that do not contain any entities. At least they should not contain any entities based on our entity selection in the rules-based approach (Chapter 3). But there are some rules that might have left some potential entities behind, meaning that in the training dataset, we may have words that are currently marked as not being an entity even though they are one.

The most concerning rule that could have caused this is the one removing all words/phrases which contain morphologically unknown words (Subsection 3.5 rule 2). Another rule that can cause a similar problem is the one saying that morphologically unknown words that have a capital lemma can be marked as entities only if they are at the beginning of a sentence (Subsection 3.3.4). Therefore, we removed all stanzas containing a word(s) with a capital lemma and are currently marked as not being an entity (regardless of their morphology).

The number of entities in Table 6.2 and the ratios in Figure 6.1 are calculated after this cleaning (the cleaning was done on both datasets).

6.1.3 Train, Dev and Test split

The training data are split into three subsets (datasets): train, dev (validation), and test dataset. However, as we can see from Table 6.1, many of the stanzas we use for training do not have any entities.

So to ensure all three subsets (train, dev, test) have the same ratio of stanzas with and without entities, we first decided to split our whole training dataset into two sets. One of these sets contains stanzas that do have at least one entity in them, and the second one contains stanzas that do not have any entities in them. Both of these sets were split into three subsets using the same ratios. The ratio for dev data is 15% of the total dataset (15% from the set containing only empty stanzas + 15% from the set containing only stanzas with entities). Then 15% of the remaining data (15% from the remains of the set containing only empty stanzas + 15% from the remains of the set containing only stanzas with entities) is used as test data, and the rest of the data as train data.

6.1.4 Input format for training data

The model requires a certain way of formatting the train, dev, and test data. Firstly, all of these datasets have to be in 3 separate *.txt* files. In each file, the format should be as in Figure 6.2. The input requires every word to be on a new line and every sentence (stanza being a “sentence” in our case) to be separated by an empty line. The required input feature is only a word token (or any form of a word on which a word embedding can be applied (in our case, it will be a word lemma)). Additional features, however, can be added as well. In the case seen in Figure 6.2, the word lemma and morphological tag is added. Lastly, the BIO label needs to be added so that the model can know which entity the word represents.

```

Position in the sentence/stanza \t Word token \t Additional features \t BIO-label
1 Petr NNMS4-----A----- Petr B-PERSON
2 je VB-S---3P-AA---I být 0
3 České AAFS2----1A----- Český B-PLACE
4 jméno NNNS4-----A----- jméno 0

1 New X24----- New 0
2 stanza X24----- stanza 0

```

■ **Figure 6.2** Example of input text format

6.2 Metrics used for evaluation

For evaluation of the model and for hyperparameter tuning, an F1 score and occasionally also accuracy and a confusion matrix are used. F1 score is implemented by default in the model that we are using. Unfortunately, we didn't find anywhere in the documentation a description of an implementation of this F1 score.

Therefore, we also decided to include an accuracy measurement as well. Three types of accuracies are being measured:

- Overall total accuracy: $\# \text{ Words with correctly predicted BIO tag} / \# \text{ All words}$
- Accuracy of predicting whether or not a given word is an entity (regardless of entity category): $\# \text{ Words correctly predicted if being an entity or not} / \# \text{ All words}$
- Accuracy of predicting correct entity category. We select only those words which have both the predicted label and the correct label marked as an entity: $\# \text{ Entities with correctly predicted entity category} / \# \text{ All entities}$

The confusion matrix is a typical confusion matrix implemented using a sklearn library. Real values are displayed in rows, and predicted values are in columns. As an example of a confusion matrix, we can see Table 6.3. If we look, for example, at the cell in row B-PERSON and column B-PLACE, we can deduce that the model labeled 154 words as an entity of Place even though they should have been labeled as an entity of Person.

■ **Table 6.3** Confusion matrix example (real values are in rows and predicted values are in columns)

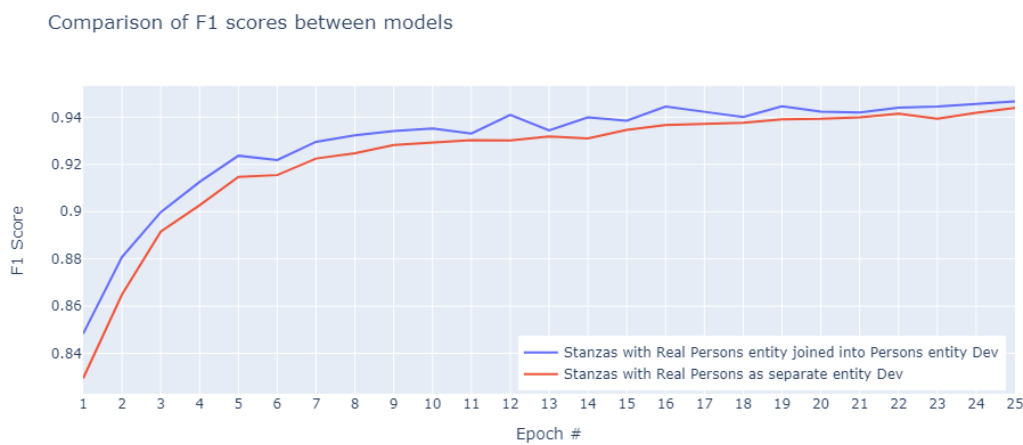
-	O	B-OTHER	B-PLACE	B-PERSON
O	1531693	0	12	69
B-OTHER	3	54	8	9
B-PLACE	22	2	3487	246
B-PERSON	19	6	154	6154

6.3 Removing Real Person as an entity category

Using our rule-based approach (Chapter 3 and 4), we separated entities into five different categories (Place, Person, Real Person, Mystic Person, and Other). However, the category of Real Person is hard to separate from the category of Person just based on the context found in the

text of the stanza itself (people who lived in real life and fictional characters are being addressed in the text the same way most of the time). To separate them, we need some sort of outside knowledge (like Wikipedia was used, for instance, in Chapter 4). Therefore, we have decided to try to remove the entities of Real Persons by labeling them just as Person entities. Theoretically, the entity category of Real Person can be later added back by using the Wikipedia approach. Plus, we believe if we have one category of a Person entity less, the model may be more effective in differentiating the rest of the entity categories.

To test the potential improvement, we trained two models, one with Real Persons and one without. The train, dev, and test datasets are the same for both models. The only difference is that the datasets used for training the model without Real Persons are now changed, so the entities that were marked as Real Persons are now marked just as Persons. Also, all of the hyperparameter values for both models are the same.



■ **Figure 6.3** Comparison between models with or without Real Person category

Accuracy of the model without Real Person entities (measured as explained in Section 6.2):

- Overall total accuracy: 0.99948
- Accuracy of predicting whether or not a given word is an entity (regardless of entity category): 0.99984
- Accuracy of predicting correct entity category. We select only those words which have both the predicted label and the correct label marked as an entity: 0.94597
- Accuracy of predicting correct entity category (Entities Person and Real Person viewed as the same category this time, so we can compare with the model which does not have Real Person entity): 0.94632

Accuracy of the model with Real Person entities (measured as explained in Section 6.2):

- Overall total accuracy: 0.99950
- Accuracy of predicting whether or not a given word is an entity (regardless of entity category): 0.99983
- Accuracy of predicting correct entity category. We select only those words which have both the predicted label and the correct label marked as an entity: 0.94746

■ **Table 6.4** Confusion matrix model without Real Person entity (predicted values are in columns and real values are in rows)

-	O	B-OTHER	B-PLACE	B-PERSON	B-MYSTIC PERSON
O	1531700	1	11	46	24
B-OTHER	3	58	6	7	0
B-PLACE	15	7	3458	265	13
B-PERSON	44	0	137	6912	21
B-MYSTIC PERSON	2	0	12	47	404

■ **Table 6.5** Confusion matrix model with Real Person entity (predicted values are in columns and real values are in rows)

-	O	B-OTHER	B-PLACE	B-PERSON	B-MYSTIC PERSON	B-REAL PERSON
O	1531678	1	16	54	10	17
B-OTHER	1	59	7	6	1	0
B-PLACE	11	3	3488	239	12	7
B-PERSON	24	4	128	6156	7	9
B-MYSTIC PERSON	3	0	19	38	405	0
B-REAL PERSON	4	1	8	63	4	702

As we can see in Figure 6.3, the model without Real Persons outperforms the model with Real Persons across all epochs even though just by a little bit (the best F1 score for the model without Real Persons is 0.9467 and for the other model it is 0.9439). From confusion matrices in Tables 6.5 and 6.4, it is hard to tell which model performs better (it almost seems that model with Real Person entities performs slightly better). The accuracy of predicting correct entity categories is slightly higher if we do not use the Real Person entities, but the difference is again minuscule. In the end, we decided to remove the category of a Real Person from the training of our model, mainly because of the reasons already mentioned above (hard to differentiate Real Persons from Persons just based on context found in a text).

6.4 Hyperparameter testing/tuning

This implementation of BiLSTM-CRF is highly configurable and has many hyperparameters that can be tuned so that the final model can be as accurate as possible. However, due to the lack of computing power (personal laptop with CPU and no dedicated GPU used), the training time for a single model (25 epochs) took around 12 hours (depending on hyperparameters).

Therefore, not all hyperparameters were tested, and the ones that were, were not always tested thoroughly. A greedy-like approach was used, which means we optimized one hyperparameter at a time since tuning multiple at once would create too many combinations of hyperparameters for me to train. Starting with tuning the hyperparameters that should have a higher impact on the final score and moving on to the lower impactful ones. Thanks to the Reimers et al. (2016) paper [20], we know which hyperparameters have a higher impact on the final F1 score (tested for NER on CoNLL 2003 (Reuters) dataset). We also know what hyperparameter values were optimal for the CoNLL 2003 (Reuters) dataset regarding NER. However, our dataset is in a different language, and it is a dataset of poems and not plain texts, which makes it a bit different.

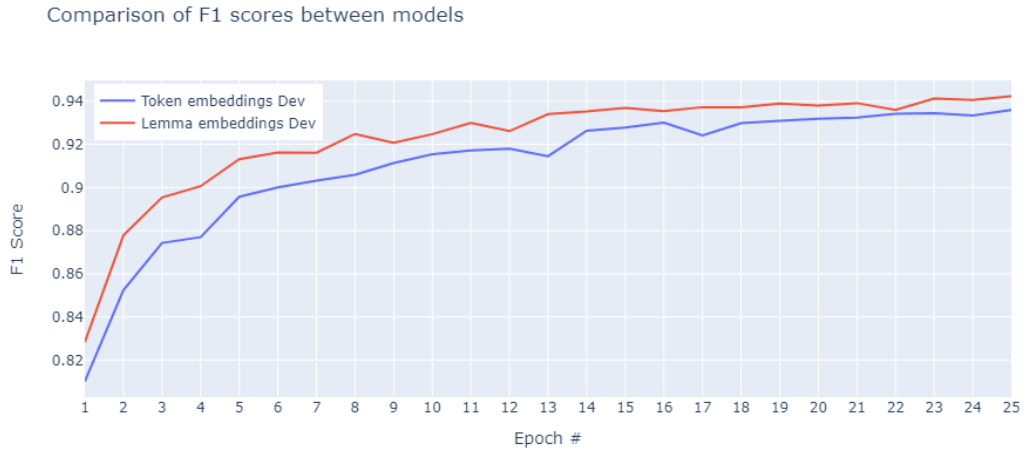
6.4.1 Word embeddings

The idea of word embedding is to transform a word from a sequence of characters into a vector. There are many types of word embeddings. The word embedding type used for our model is called word2vec. It was chosen thanks to its free, trainable, and easy-to-use API implementation done by Gensim.

Although, according to Dařena et al. (2020), the best word embedding for NER would be fastText, it was still followed closely by word2vec in many of their tests, and word2vec still performed better than GloVe in their work. [25] Similar findings were made by Hořeňovká (2019) since, in the conclusion of her work, she stated: “GloVe model using the default parameter settings does not seem to work well on Czech, that CBOW architecture of Word2Vec/fastText generally outperforms the Skip-gram architecture (unlike on English) and that LexVec performs fairly well in our experiments.” [26]

The trickier question was whether to train and use word embeddings on the token format of the words or the lemma format of the words. Hořeňovká (2019) finding recommends training models on lemmatized corpus for NLP tasks done in the Czech language. [26]

Therefore, we decided to train two embeddings, one on tokens and one on lemmas. The embeddings were trained on the Corpus of the Czech Verse (Chapter 2). Then both of these embeddings were used as inputs to train a model. One model was trained on word tokens (using token embedding) and another on word lemmas (using lemma embedding). The results are seen in Figure 6.4 and in Tables 6.6 and 6.7.



■ **Figure 6.4** Lemma vs. Token embedding

■ **Table 6.6** Confusion matrix for the model with lemma embedding (predicted values are in columns and real values are in rows)

-	O	B-OTHER	B-PLACE	B-PERSON	B-MYSTIC PERSON
O	1531684	1	18	51	24
B-OTHER	2	59	2	11	0
B-PLACE	18	4	3404	327	6
B-PERSON	38	0	127	6934	10
B-MYSTIC PERSON	2	0	10	58	395

■ **Table 6.7** Confusion matrix for the model with token embedding (predicted values are in columns and real values are in rows)

-	O	B-OTHER	B-PLACE	B-PERSON	B-MYSTIC PERSON
O	1484315	540	17799	27895	1205
B-OTHER	6	33	9	24	0
B-PLACE	192	5	2739	803	7
B-PERSON	197	13	914	5925	40
B-MYSTIC PERSON	36	0	73	114	241

Figure 6.4 shows that the model trained on lemma embeddings outperformed the one trained on token embeddings, although only by 0.0064 in the best F1 Score. The confusion matrices also tell us that the model trained on lemma embedding seems to be superior of those two. The question is, what if we used both embeddings? This question is tested in the next subsection (Subsection 6.4.2).

6.4.2 Input features

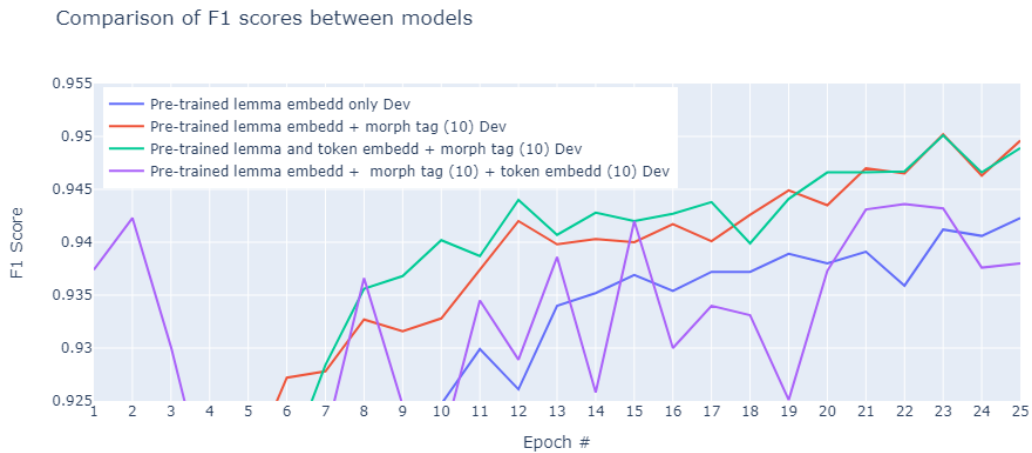
Until now, the models presented in this thesis have been trained only using the lemmas (or tokens) of each word as an input feature. The BiLSTM model that we use allows for the usage of multiple input features. Therefore, we tried to add word tokens and morphological tags as input features.

By default, any other extra feature, apart from lemmas that have their own pre-trained embedding (created by us), has embeddings created by a class called `Embedding` from a package called `keras.layer`. Unfortunately, we could not figure out from the Keras documentation how exactly these extra features are being converted to embeddings. Especially since the Keras documentation for this `Embedding` class does not contain some of the arguments that are used in the `Embedding` class of the implementation of BiLSTM-CRF that we are using (the `Embeddings` class in our model, for example, uses argument `weights` which is not present in the Keras documentation).

Even though we do not know the exact detail of how the initialization of some of these extra features works, we can still add them as input features. So the model does not only take the lemma of a word as an input but also its morphological tag and the token form of the word as an input. These extra features (morphological tag and token in this case) need a dimension size for their embeddings as a hyperparameter. We started with size 40 and optimized the size of the dimension later on (Subsection 6.4.3). It is good to note that the dimension size of the extra feature inputs has to be the same for all of these features (restriction of the model implementation).

We could also add the token format of words as a pre-trained embedding (adding it the same way as lemma formats of words are added). So both lemma and token pre-trained embeddings (pre-trained in the way as said in Subsection 6.4.1) are used in one of the tested models. However, to use two pre-trained embeddings, the model needed a slight change in its code since the model was created for the usage of only one pre-trained embedding.

So overall, to test different feature inputs, four models were trained. One with pre-trained lemma embedding and morphological tag with dimensional size 40. One with pre-trained lemma embedding and also pre-trained token embedding and morphological tag with dimensional size 40. The third one is with pre-trained lemma embedding and both word token and morphological tag as input features that are embedded using the Keras Embedding class (with size 40 for both). And last one is a model trained on pre-trained lemma embedding only as a control (since lemma embedding was shown to outperform token embedding in Subsection 6.4.1).



■ **Figure 6.5** Input features (zoomed y axis so the difference between F1 scores can be seen better)

As we can see from Figure 6.5, the F1 scores for the model with pre-trained lemma embedding and morphological tag perform very similarly to the model with pre-trained lemma and token embedding and morphological tag. The difference in their best F1 scores was only 0.0001, which is minuscule.

However, if we look at the confusion matrices for these two models (Table 6.8 and 6.9), we can see that the model containing pre-trained token embedding as well as the lemma embedding predicts better entities of place and, on the other hand, the model with pre-trained lemma embedding only, predicts entities of a person better. Since the results are both very similar, we have decided to carry on tuning the model with lemma embedding and morphological tag only since it is simpler, so it takes less time to train it. Therefore, we can tune more additional hyperparameters on it.

6.4.3 Embedding dimension size for morphological tag

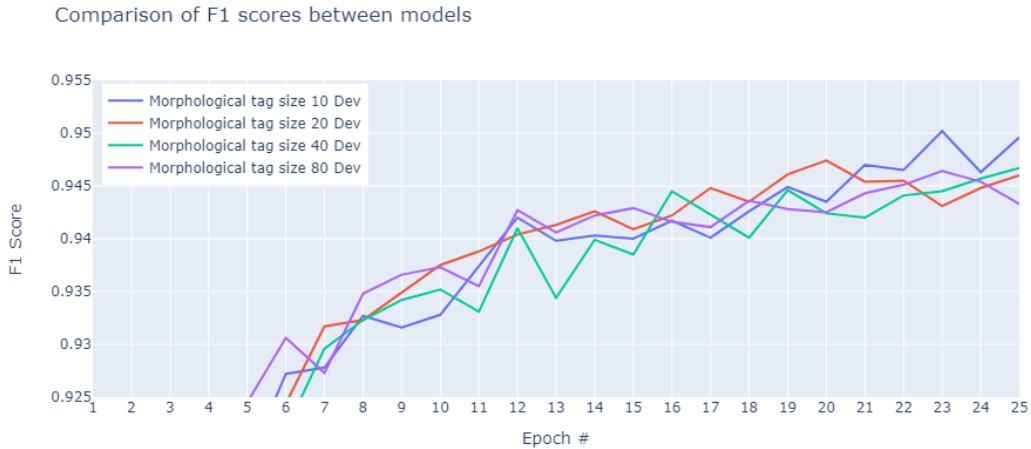
So far, the best model we found uses the morphological tag of a word as an extra input feature. Our model allows us to tune the size of the dimension for the embedding, into which the morphological tag is transformed. We tested four different sizes of dimensions: 10, 20, 40, and 80 (for reference, our pre-trained lemma embedding uses a size of 100).

■ **Table 6.8** Confusion matrix for the model with lemma embedding and morphological tag feature (predicted values are in columns and real values are in rows)

-	O	B-OTHER	B-PLACE	B-PERSON	B-MYSTIC PERSON
O	1531726	0	15	40	9
B-OTHER	2	62	3	7	0
B-PLACE	12	5	3473	263	8
B-PERSON	32	1	124	6942	8
B-MYSTIC PERSON	2	0	11	54	398

■ **Table 6.9** Confusion matrix for the model with pre-trained lemma and token embedding and morphological tag feature (predicted values are in columns and real values are in rows)

-	O	B-OTHER	B-PLACE	B-PERSON	B-MYSTIC PERSON
O	1531764	0	11	30	5
B-OTHER	2	59	4	8	1
B-PLACE	18	4	3493	229	13
B-PERSON	19	2	173	6905	15
B-MYSTIC PERSON	2	0	11	43	409



■ **Figure 6.6** Different morphological tag sizes of embedding (zoomed y axis so the difference between F1 scores can be seen better)

As we can see from Figure 6.6, the differences are quite minor, but the best F1 score was achieved with a dimensional size of 10 for the embedding of the morphological tag.

6.4.4 Hyperparameter tuning

The model has many hyperparameters that can be tuned. Based on the recommendation of Reimers et al. (2017), who are the authors of this model, the hyperparameters that have the highest effect on the overall F1 score in NER are: word embeddings (already tuned in Subsection 6.4.1), optimizer, gradient normalization, classifier, and dropout. Apart from these “high” impact hyperparameters, as they are called in Reimers et al. (2017), we also tuned #LSTM layers which is said to have “medium” impact. [20] The remaining model hyperparameters that we did not tune are: character representation, gradient clipping, tagging scheme, recurrent units, mini-batch size, and backend. Impact on the F1 score by all of these remaining hyperparameters is said to be “medium” or “low” by Reimers et al. (2017) [20], so due to time constraints, we did not tune these hyperparameters.

The hyperparameter values that are seen in Table 6.10 were tuned. It is important to note that we only tuned one hyperparameter at a time while the rest of the hyperparameters were set to their default values by the model (the default values are the ones that were found to give the best result for NER in Reimers et al. (2017) [20]).

So, as we can see from Table 6.10, we could not find better hyperparameters than the ones set by default. The closest F1 score to the one set by the default hyperparameters was formed by a model using Adam optimizer instead of Nadam (difference of 0.0013 in F1 score).

¹Variational dropout is a kind of dropout applied to both the output and recurrent units of the LSTM networks. [20] In the brackets in Table 6.10, the first number corresponds to the output dropout and the second number to the recurrent dropout. [18]

■ **Table 6.10** Best hyperparameter values

Hyperparameter name	Tested hyperparameter values	Best hyperparameter value (F1 score rated)
Optimizer	Nadam (default), Adam, RMSProp	Nadam
Gradient Normalization	1 (default), 5	1
Classifier	CRF (default), Softmax	CRF
Dropout	Variational (0.25, 0.25) (default), Variational (0.15, 0.15), Variational (0.05, 0.05) ¹	Variational (0.25, 0.25)
#LSTM Layers	2 (default), 3	2

6.5 Final model

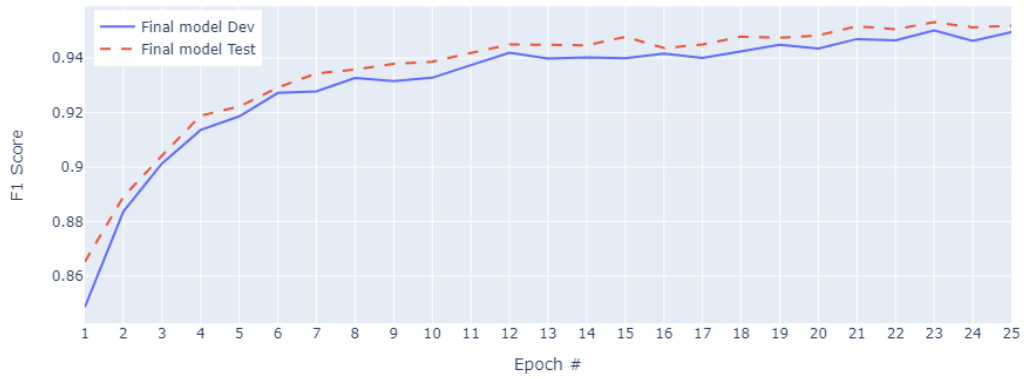
The best model we found (best F1 score and accuracy) is trained on all stanzas in the CCV dataset, which had all their entities fully categorized. The data were split into train(72.25%), dev(15%), and test(12.75%) datasets. Real Person entities were converted into Person entities, and word2vec word embedding trained on word lemmas is used instead of embedding trained on tokens. The input features chosen for this model are word lemmas (using our pre-trained lemma embedding) and morphological tags (embedded using the Embedding class from Keras with a dimensional size of 10). The hyperparameters for this model are mentioned in Table 6.10.

The F1 score of this model for the test dataset is 0.9532, and its development across all epochs can be seen in Figure 6.7. The confusion matrix of this model for the test dataset can be seen in Table 6.11. And the accuracies of this model measured on the test dataset are the following (measured as explained in Section 6.2):

- Overall total accuracy: 0.99904
 - Total words: 1,810,983
 - Correctly predicted words: 1,809,244
- Accuracy of predicting whether or not a given word is an entity (regardless of entity category): 0.99983
 - Total words: 1,810,983
 - Correctly predicted words: 1,810,677
- Accuracy of predicting correct entity category. We select only those words which have both the predicted label and the correct label marked as an entity: 0.88870
 - Total entities: 13,720
 - Correctly predicted entities: 12,193

It is important to note that the so-called real values that the F1 score, accuracy, and confusion matrix use are actually values created by our rule-based approach. Therefore, these values contain some flows since no rule-based system can be 100% accurate for selecting entities (although we hope that the number of mistakes in the training dataset is minimal).

Comparison of F1 scores between models



■ **Figure 6.7** Final model F1 score (Dev and Test datasets)

■ **Table 6.11** Confusion matrix for the final model created on test dataset (predicted values are in columns and real values are in rows)

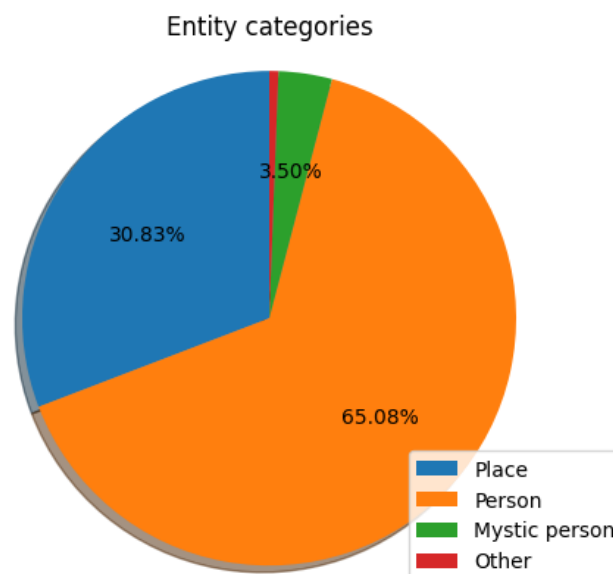
-	O	B-OTHER	B-PLACE	B-PERSON	B-MYSTIC PERSON
O	1531726	0	15	40	9
B-OTHER	2	62	3	7	0
B-PLACE	12	5	3473	263	8
B-PERSON	32	1	124	6942	8
B-MYSTIC PERSON	2	0	11	54	398

6.5.1 Results of final model

After creating the final model (Section 6.5), we ran it on all of the books from the CCV. In Table 6.12, we can see how many entities of each kind and how many in total we found in the CCV, and in Figure 6.8, we can better see the proportion of each entity category in the CCV dataset. So the most common entities in the CCV are those of a Person followed by entities of a Place, with only a small amount of entities being either entities of a Mystic Person or Other.

■ **Table 6.12** Number of entities in the whole CCV

Entity category	Number of entities
Person	130,589
Place	61,860
Mystic Person	7,017
Other	1,190
Total number of entities	200,656
Total number of words	15,773,641



■ **Figure 6.8** Entity categories distribution

6.5.2 Saving and visualizing results

The result of running the final model on the CCV was saved into *json* files (one file for each book in the CCV) with the same scheme in each file as it was in the CCV dataset (seen in Picture 2.1). The only difference is that we added a key called *'entity'* into the *'word'* dictionary. So each word in the resulting dataset will now hold information about its entity category. For labeling which entity each word holds, BIO tagging is used (Subsection 1.4.1). An example of the resulting dataset can be seen in Figure 6.9.

```

"text": "(pan Viktor Dyk cos, tu\u0161\u00eddm, sly\u0161el taky).",
"punct": {
  "5": ",",
  "4": ",",
  "7": ").",
  "0": "("
},
"words": [
  {
    "token_lc": "pan",
    "xsampa": "pan",
    "morph": "NNMS1-----A-----",
    "phoebe": "pan",
    "token": "pan",
    "lemma": "pan",
    "entity": "0"
  },
  {
    "token_lc": "viktor",
    "xsampa": "vIkto\u0159",
    "morph": "NNMS1-----A-----",
    "phoebe": "vikto\u0159",
    "token": "Vikto\u0159",
    "lemma": "Vikto\u0159",
    "entity": "B-PERSON"
  },
  {
    "token_lc": "dyk",
    "xsampa": "dIk",
    "morph": "NNMS1-----A-----",
    "phoebe": "dik",
    "token": "Dyk",
    "lemma": "Dyk",
    "entity": "I-PERSON"
  },
  {
    "token_lc": "cos",
    "xsampa": "t_sos",
    "morph": "NN-----8-",
    "phoebe": "cos",
    "token": "cos",
    "lemma": "cos",
    "entity": "0"
  }
]

```

■ Figure 6.9 Resulting schema example

For easier visualization, a set of *html* templates was also created. Each template contains text from one book from the CCV and has colorfully marked entities (each color for a different category). An example of this *html* template can be seen in Figure 6.10.

Již uhašena hlaveň **Lojology** dumy ,
co z vlasti které zbylé dožírala rummy .
Již ostal **Žižka** mnišské troskotati sbory
a netřeba se krýti v lesích před **Tábory** .

■ **Figure 6.10** Visualization example

Conclusion

Conclusion of this bachelor thesis.

We have created a program that recognizes and finds entities of Person, Mystic Person, Place, and Other in the CCV (CCV description in Chapter 2). Firstly we created a training dataset using a rule-based approach (Chapters 3 and 4) and then used this training dataset to train a BiLSTM-CRF neural network (Chapters 5 and 6).

Since the training data were not created by labeling words as entities by hand, there are already a few mistakes in the training dataset. Therefore it is hard to estimate the exact accuracy of the prediction of named entities. The accuracies, F1 scores, and confusion matrices in this thesis are created under the assumption that the training dataset was 100% accurate.

7.1 Room for improvement

Ways to improve the final result could be some changes in the rule-based approach used for selecting training data. For example, in the rule-based approach, when selecting entities (Chapter 3), we rely on the capitalization of the word lemmas. But the model made by Morphodita [14], which lemmatized the CCV, was probably not tested much on the fact of whether or not a given word lemma should be capitalized but rather on the fact if the word lemma is in the correct form. So instead, we maybe should have focused on word tokens more instead (even though the rule-based approach using them seemed to produce worse results at first, as discussed in Subsection 3.3.1).

In the rule-based approach, when categorizing entities using Wikipedia (Chapter 4), we could have been more restrictive (categorize entities only if we are more certain about the category it represents). The problem with being too restrictive was that we would probably not have categorized enough entities to create a large enough training dataset. But in the end, we categorized enough entities to use 94% of the total number of stanzas in the CCV as our training data. So in retrospect, we could have been more restrictive.

Lastly, we could have also tried to use other models than just BiLSTM-CRF, but this was not done due to time constraints.

Bibliography

1. *Corpus of Czech Verse* [online]. 2023. [visited on 2023-03-29]. Available from: https://versologie.cz/v2/web_content/corpus.php?lang=en.
2. JAGOTA, Arun. *Named entity recognition in NLP* [online]. Towards Data Science, 2020-10 [visited on 2023-03-15]. Available from: <https://towardsdatascience.com/named-entity-recognition-in-nlp-be09139fa7b8>.
3. *A Comprehensive Guide to Named Entity Recognition (NER)* [online]. Turing [visited on 2023-04-20]. Available from: <https://www.turing.com/kb/a-comprehensive-guide-to-named-entity-recognition>.
4. *Uncover hidden insights: Advanced named entity recognition* [online]. 2023-03. [visited on 2023-04-20]. Available from: <https://www.expressanalytics.com/blog/what-is-named-entity-recognition-ner-benefits-use-cases-algorithms/>.
5. KEVIN, Vu. *BERT Transformers: How Do They Work?* [online]. DZone, 2021-04 [visited on 2023-04-20]. Available from: <https://dzone.com/articles/bert-transformers-how-do-they-work>.
6. CARIELLO, Maria Carmela; LENCI, Alessandro; MITKOV, Ruslan. A comparison between named entity recognition models in the biomedical domain. *Proceedings of the Translation and Interpreting Technology Online Conference TRITON 2021* [online]. 2021, pp. 76–84 [visited on 2023-04-20]. Available from DOI: 10.26615/978-954-452-071-7_009.
7. MARSHALL, Christopher. *What is named entity recognition (NER) and how can I use it?* super.AI, 2020-06. Available also from: <https://medium.com/mysuperaai/what-is-named-entity-recognition-ner-and-how-can-i-use-it-2b68cf6f545d>.
8. *Named entity recognition* [online]. 2021. [visited on 2023-03-15]. Available from: https://natural-language-understanding.fandom.com/wiki/Named_entity_recognition#cite_ref-Ratinov_2009_1-0.
9. RATINOV, Lev; ROTH, Dan. Design Challenges and Misconceptions in Named Entity Recognition. In: *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*. Boulder, Colorado: Association for Computational Linguistics, 2009, pp. 147–155. Available also from: <https://aclanthology.org/W09-1119>.
10. *Corpus of Czech Verse, PHoEBE phonetic transcription* [online]. 2022. [visited on 2023-04-20]. Available from: https://versologie.cz/v2/web_content/phoebe.php.
11. *SAMPA / X-SAMPA Transcription of Czech* [online]. 2015. [visited on 2023-04-20]. Available from: <https://fonetika.ff.cuni.cz/o-fonetice/foneticka-transkripce/czech-sampa/>.
12. PLECHÁČ, Petr; ZOUHAR, Vilém. *Versotym/Corpusczechverse* [online]. GitHub, 2021 [visited on 2023-03-29]. Available from: <https://github.com/versotym/corpusCzechVerse>.

13. MIKULOVÁ, Marie; HAJIČ, Jan; HANA, Jiří; HANOVÁ, Hana; HLAVÁČOVÁ, Jaroslava; JERÁBEK, Emil; ŠTĚPÁNKOVÁ, Barbora; HLADKÁ, Barbora Vidová; ZEMAN, Daniel. Manual for Morphological Annotation. Revision for Prague Dependency Treebank – Consolidated 2020 release. In: *Manual for Morphological Annotation Revision for Prague Dependency Treebank – Consolidated 2020 release*. Malostranské nám. 25, CZ-11800 Prague 1, Czechia: Institute of Formal and Applied Linguistics (ÚFAL MFF UK), 2020. Available also from: <https://ufal.mff.cuni.cz/techrep/tr64.pdf>.
14. STRAKOVÁ, Jana; STRAKA, Milan; HAJIČ, Jan. 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*. Baltimore, Maryland: Association for Computational Linguistics, 2014, pp. 13–18. Available also from: <http://www.aclweb.org/anthology/P/P14/P14-5003.pdf>.
15. BOHÁČ, Pavel; HARVALÍK, Milan. *Toponymic Guidelines of the Czech Republic*. CZ-182 11 Praha 8, Pod sídlíštěm 9: Český úřad zeměměřický a katastrální, 2004. ISBN 80-902321-0-8. Available also from: https://www.cuzk.cz/Zivotni-situace/Poradci-a-poradni-organy/Nazvoslovna-komise-CUZK/Geograficke-nazvoslovne-seznamy/Toponymic_guidelines_of_the_Czech_republic.aspx.
16. *Help:Categories - Wikipedia* [online]. Wikimedia Foundation, 2021 [visited on 2023-04-15]. Available from: <https://en.wikipedia.org/wiki/Help:Categories>.
17. *Help:Category - Wikipedia* [online]. Wikimedia Foundation, 2021 [visited on 2023-04-15]. Available from: <https://en.wikipedia.org/wiki/Wikipedia:Category>.
18. UKPLAB. *emnlp2017-bilstm-cnn-crf: BiLSTM-CNN-CRF architecture for sequence tagging* [online]. GitHub, 2018 [visited on 2023-05-01]. Available from: <https://github.com/UKPLab/emnlp2017-bilstm-cnn-crf>.
19. REIMERS, Nils; GUREVYCH, Iryna. *Reporting Score Distributions Makes a Difference: Performance Study of LSTM-networks for Sequence Tagging* [online]. 2017. [visited on 2023-05-09]. Available from arXiv: 1707.09861 [cs.CL].
20. REIMERS, Nils; GUREVYCH, Iryna. Optimal Hyperparameters for Deep LSTM-Networks for Sequence Labeling Tasks. *CoRR*. 2017, vol. abs/1707.06799. Available from arXiv: 1707.06799.
21. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. Cambridge, MA: MIT Press, 2016. ISBN 978-0262035613.
22. LIPTON, Zachary C.; BERKOWITZ, John; ELKAN, Charles. *A Critical Review of Recurrent Neural Networks for Sequence Learning*. 2015. Available from arXiv: 1506.00019 [cs.LG].
23. HUANG, Zhiheng; XU, Wei; YU, Kai. Bidirectional LSTM-CRF Models for Sequence Tagging. 2015. Available from DOI: 10.48550/ARXIV.1508.01991.
24. LAFFERTY, John D.; MCCALLUM, Andrew; PEREIRA, Fernando C. N. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 282–289. ICML '01. ISBN 1558607781.
25. DAŘENA, František; SÜSS, Michael. Quality of Word Vectors and its Impact on Named Entity Recognition in Czech. *European Journal of Business Science and Technology*. 2020, vol. 6, pp. 154–169.
26. HOŘEŇOVSKÁ, Karolína. An evaluation of Czech word embeddings. In: *Proceedings of the 22nd Nordic Conference on Computational Linguistics*. Turku, Finland: Linköping University Electronic Press, 2019, pp. 65–75. Available also from: <https://aclanthology.org/W19-6107>.

Source code

The source code can be found on the GitLab repository called `NER_poezie_cernyo14` (https://gitlab.fit.cvut.cz/cernyo14/ner_poezie_cernyo14).