SysNERV
Technical Documentation

Jindřich Helcl, Petr Jankovský, Ondřej Košarko,
Jan Mašek, Sara van de Moosdijk

supervisor: Zdeněk Žabokrtský

June 5, 2013

Contents

Chapter 1

Introduction

In this document we present SysNERV, a system for named entity recognition and visualization. It is a collection of tools which includes a web service with named entity recognition, a Chrome browser extension which uses that web service, and a stand-alone named entity recognition application for offline use.

In this introductory chapter we will explain the basics of named entity recognition, a brief history of the field, and the technologies we used, such as Support Vector Machine (SVM) modeling and a Chrome browser extension. We introduce Treex, an NLP[1] framework which SysNERV is a part of, and we describe the need for a web service to connect Treex with the browser extension. Each section will give a brief overview of the system, with more detailed descriptions in the consecutive chapters of this document.

1.1 Named Entity Recognition

Named entity recognition is a field of Natural Language Processing (NLP) aimed at finding named entities in given text. For a better understanding of what that entails, one must define what the term *named entity* means. Nadeau and Sekine (2007) [9] state that the first word *Named* in the term

[1] Natural Language Processing

suggests that there are some names involved. This is step in the right direction, but does not yet cover the entire definition. According to their article, a named entity is a word or a phrase that refers to one particular and fixed thing. For example, *the capital city of the Czech Republic* is referred to by the named entity *Prague*. There is also a general consensus to include some temporal and numerical values in the category of named entities. These are compliant with the definition of being a referrer since you can mark the year *2013* as an entity referring to the thirteenth year of the third millennium. Of course, not all temporal and numerical values are valid occurrences of named entities. Look for example at the word *Monday*. Our definition suggests that *Monday* is not a named entity since it may refer to many different Mondays and we do not know which one to pick. On the other hand, Monday *is* a name, so it would have been a named entity based on the very first definition alone. Since the definition of named entities can very slightly, we will define the exact categories used by our NER in a later chapter.

A Brief History of Named Entity Recognition

One important article for this subject is the article by Nadeau and Sekine (2007) already mentioned in the previous section. They made a survey of fifteen years of research in the field and explored research in different aspects of the field.

But one of the first papers in this field was presented by Lisa F. Rau (1991) [10] and was dedicated to the automatic extracting of company names from text. Since then there has been much progress in performing the task in many different languages as well as in specializing the task for different domains (subsets of a language used in particular contexts, e.g. technical reports, newspapers). It is a known fact that machine learning models in NLP which are restricted to specific domains perform better than those trained on general texts in a language with no additional assumptions.

The collection of entity types has also gone through certain developments. It started with the proper names of people, locations and organizations such as *John*, *New York* or *Charles University*. Miscellaneous entity types were added later. These include entities like telephone numbers, e-mail addresses, and book titles. There are also domain-specific entity types, like proteins or

chemicals. Our exact entity type set is listed and described below, in section 2.2.

Throughout history there have been many machine learning methods applied to performing the NER task. For supervised learning (when annotated training data was available), Hidden Markov Models (Bikel et al., 1997) [2], Decision Trees (Sekine, 1988) [11], Maximum Entropy Models (Borthwick, 1998) [3], Conditional Random Fields (McCallum and Li, 2003) [8] and Support Vector Machines (SVM) (Asahara and Matsumoto, 2003) [1] have all been used. In our application, we use an SVM model. For a more detailed introduction to this kind of modelling see section 1.2 below.

Besides supervised learning methods, there were also methods called "semi-supervised" or completely unsupervised methods (which means there is minimum to no training data provided at all). These methods are mainly based on clustering. You can see a further description of these methods in (Nadeau and Sekine, 2007).

In that article one can also see different feature sets[2] for different learning methods. Many of those features depend on the token itself[3], its morphology (the way the word is formed from its base form), and its orthography (the way the token is spelled). Other features may include its presence in a list of known names, syntax-based features (coreference), document meta information, and so on. Our feature set is described in section 2.4

1.2 Support Vector Machines

Support Vector Machines are a non-probabilistic approach to supervised machine learning tasks. The current standard version of the algorithm was proposed by (Vapnik and Cortes, 1995) [5]. A binary classification task means to find a function that separates two different types of observations (e.g. those words that are named entities and those that are not). These observation types are commonly called *classes* or *labels*.

There are no arbitrary assumptions on observations themselves – they can

[2]See section 1.2 for information on what features are.

[3]Tokens are mainly words, but other things such as punctuation marks or numbers are considered tokens as well.

generally be any real-world objects. However, when we work with real-world objects in computer science, we have to represent them formally, in a way computers understand. Also, we want to prevent creating an impossible task, so we have to incorporate some simplification. We introduce object attributes and represent the object using a vector of values with manually selected attributes of the object. These attributes are also known as *features*.

1.3 Treex

As stated on the Treex webpage [12],

> Treex (formerly TectoMT) is a highly modular NLP software system implemented in Perl programming language under Linux. It is primarily aimed at Machine Translation, making use of the ideas and technology created during the Prague Dependency Treebank project. At the same time, it is also hoped to significantly facilitate and accelerate development of software solutions of many other NLP tasks, especially due to re-usability of the numerous integrated processing modules (called blocks), which are equipped with uniform object-oriented interfaces.

Further information on Treex and its use for our application can be found in section 2.1.

1.4 Web service

The named entity recognizer should be easy to use and access whenever reading a text online. Users should be able to use the recognizer on any webpage and additionally receive basic information about the entities it finds. This requires a web service to host the recognizer and retrieve the relevant information. The web service should be hosted in such a way that users need only download a simple wrapper program to access it. For bilingual users it should be able to dstinguish any English and Czech texts they submit, and retrieve basic information in the same language as the input. All of this needs to be done in a reasonable time frame. Chapter 3 provides a more

detailed account of the framework used to set up the web service and the different components required to ensure a smooth connection between the wrapper program and the named entity recognizer.

1.5 Browser extension

The wrapper program should be integrable with an internet browser and easy for the user to download , install, and use. This can be achieved by implementing a browser extension that can be activated in any tab of the browser on any piece of text. Browser extensions are very suitable for this task because their use is widespread and popular among internet users, and many browsers have a special platform for the distribution of their extensions. At the moment we are not aware of any existing extensions capable of recognizing named entities in both English and Czech, nor have we found any browser applications which focus on extracting additional information about the entities for presenting to the user. Having an extension directly connected to the browser will allow the user to utilize the named entity recognizer without having to copy-paste the text into a separate software program. Creating such a browser extension requires making some important decisions, such as which browser to use, the general design of extension, and the method of distribution. The research and reasoning behind our final decisions on all of these issues is further explained in chapter 4.

Chapter 2

Recognizer Implementation

This chapter contains all the details relating to the NER implementation and is divided into two sections. In the first section, the architecture of our NER system is described, and the second one deals with the features used by the recognizer.

2.1 Architecture

In this section, the architecture of the recognizer itself is first described, followed by a few paragraphs on the Treex architecture and a description of our system containing the recognizer, a web service, and a browser extension.

Recognizer architecture

Named entity recognition can be divided into two parts – detection of the named entity itself and recognition of its type. Recognition of named entity type is a classic classification task. Detection of a named entity means finding a sequence of tokens that suits the definition expressed earlier in this document. This sequence is then labeled by one of the defined categories or subtypes (personal name, geographical entity, etc).

The sequence can be arbitrarily long, limited only by the sentence boundary.

We can simplify this task by limiting the length of the entities we are trying to recognize. Using this trick, we restrict the problem of named entity detection to a classification task. However, with this simplification we cannot recognize named entities longer than the specified limit. We decided to set the limit on named entity length at three words since one-word, two-word and three-word named entities cover the 97.52% of all named entities in the Czech Named Entity Corpus.

Classification is based on monitoring selected features for unigrams (tokens), bigrams and sometimes trigrams. [1] Features are functions from a set of tokens to any arbitrary finite set – in other words it is whatever we can observe about the token that can be automatically evaluated, or that acquires numerical values. Based on the set of features, an external classifier determines the class to which the assessed named entity belongs. We use libsvm as the implementation of a Support Vector Machine classifier [4]. More precisely, we use Perl bindings for the mentioned library, called `Algorithm::SVM`. [7]

The whole SVM classifier building process consists of these steps:

- Feature selection
- Training of the model
- Parameter tuning
- Re-training of the recognition model with tuned parameters

In the feature selection step, we define features which will help the classifier to determine whether the token is a named entity or not. (For example – does the token start with a capital letter?) Then we train a model based on the training data. The classifier uses the model to determine which class the current token sequence belongs to. In the next step we tune the parameters *gamma* and C which alter the behavior of the SVM classifier. Correctly setting these parameters is important and can improve the results significantly. You can read more about parameter tuning in section 2.5. In the last step we re-train the recognition model with the best parameters.

[1] By bigram and trigram we mean a pair or a triplet of consecutive tokens in the sentence respectively.

Treex architecture

Treex contains many modules typically used in many different NLP tasks. Using these modules can significantly accelerate development of specific software. This is the reason why we add our module (in Treex called *block*) for named entity recognition to Treex.

The base Treex block collection and functionality is available at CPAN as `Treex::Core`. [15]

Data in Treex has a specific hierarchy. The smallest independently storable unit is a *document* consisting of a set of *document zones* and a sequence of *bundles*. A document zone contains a processed text, and bundles contain a set of *bundle zones*. These contain one sentence and its linguistic analysis for each layer of analysis. Available layers are:

- **a-layer** – analytical layer
- **t-layer** – tectogrammatical layer
- **p-layer** – phrase-structure layer
- **n-layer** – named entity layer

Each layer representation has the form of a tree, represented by the tree's root *node*. A node contains a set of specific attributes for the layer it belongs to. Each node (except the root node) has a parent node. The abstract class `Treex::Core::Node` defines common functionality for all types of nodes. Specific functionality for each linguistic layer is implemented in derived classes, `Treex::Core:Node::A`, `Treex::Core::Node::T`, `Treex::Core::Node::P`, and `Treex::Core::Node::N`. The processing units are:

- **block**
- **scenario**

Block is the smallest processing unit applicable to Treex documents. Each block corresponds to a Perl module. *Scenario* is a sequence of blocks. Blocks from a scenario are applied consecutively to a Treex document.

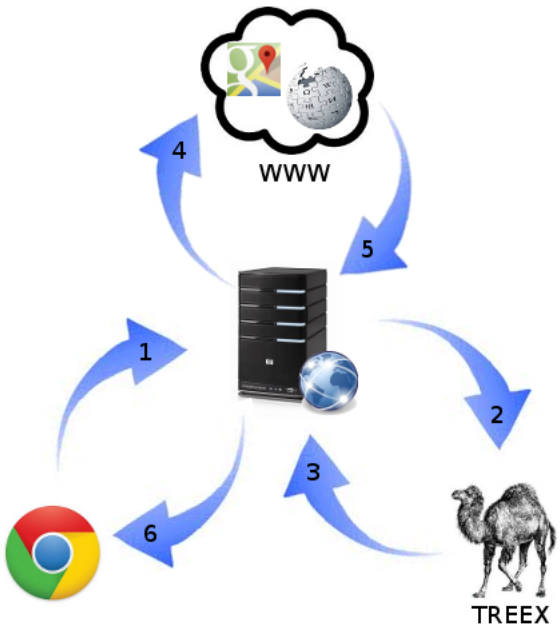

Figure 2.1: Recognizer-Web service-Extension architecture schema.

Recognizer – Web service – Extension architecture

Our work also includes a graphical user interface in the form of an extension for the web browser Google Chrome or its open-source equivalent Chromium.

The text selected for named entity recognition in the web browser is sent by the extension to the web service, which runs the Treex recognizer. For the defined named entity types it finds additional information from the web, such as a map of a geographical entity or an abstract from Wikipedia. The results are sent back to the web browser extension which displays the results. This schema is captured in Figure 2.1.

2.2 Named Entity Classification

English Classification

The Stanford NER (or CRF Classifier) used for the recognition of NEs in English texts has a one-level classification system with 4 classes: Location, Person, Organization, and Misc.

Czech Classification

Our Czech NER uses the same two-level NE classification [14] as the Czech Named Entities Corpus. The *type* of the NE is a tag composed of two characters. Table 2.1 shows all the possible tag types. Several terms are used in addition to the *type* of the NE. The *span* of the entity is the sequence of tokens of which it is composed; the beginning of the entity is marked by '<' and its end is marked by a '>'. The *container* is comprised of several NEs and possibly other words; its tag is a single uppercase letter, (see Table 2.2), and its span is marked in the same way as the spans of NEs. The *supertype* is determined by the first character of the tag. Apart from the NE types and containers, there are also additional tags used in annotation for marking eg. abbreviations, foreign words, errors in capitalization etc. These other tags are shown in table 2.3. (The examples in tables 2.1, 2.2 and 2.3 are taken from [14].)

Tag	Meaning	Examples
a	Numbers in addresses	
ah	Street number	`ul. Šikmá <ah 356>`
at	Phone/Fax number	`tel. <at 256 458 588>`
az	Zip code	`<az 180 00> Praha 8`
c	Bibliographic items	
cb	Volume number	`Mluvnice češtiny <cb 2>`
cn	Chapter/Section/Figure number	`viz obr. <cn 15>`
cp	Page number	`na straně <cp 3>`
cr	Legislative act number	`zákon č. <cr 586>`
cs	Article title	`v článku <cs On pronouns>`

Tag	Meaning	Examples
g	GEOGRAPHICAL NAMES	
gc	State	<gc Česká republika>, <gc Svatá Říše římská>
gh	Hydronym	<gh Vltava>, <gh Balaton>
gl	Nature area/object	<gl Sibiř>, <gl Apeninský poloostrov>, <gl Polabí>
gp	Planet/Cosmic object	<gp Země>
gq	Urban part	<gq Smíchov>
gr	Territorial name	<gr Morava>, <gr Rychnovsko>, <gr Badensko-Württembersko>
gs	Street/Square	<gs ul. Šikmá>, <gs nám. Míru>
gt	Continent	<gt Jižní Amerika>
gu	City/Town	<gu Praha>, <gu Opočno>
g_	Underspecified	Učkuduk v <g Mojunkumech>
i	INSTITUTIONS	
ia	Conference/Contest	<ia Stanley Cup>
ic	Cult./Educ./Scient. institution	kino <ic Dlabačov>, hokejisté <ic Sparty>
if	Company/Concern/...	<if Unipetrol>, řetězec <if Edeka>
io	Government / Political institution	<io Evropská komise>, <io Policie>, <io ODS>
i_	Underspecified	<i Studijní odd.>
m	MEDIA NAMES	
mi	Internet link	<mi www.cinestar.cz>
mn	Periodical	agentura <mn Interfax>
mr	Radio station	<mr Frekvence 1>
mt	TV station	<mt ČT 2>
n	SPECIFIC NUMBER USAGE	
na	Age	je mu <na 18>
nc	Sport score	vyhráli <nc 5:0>
ni	Itemizer	<ni 1)> DPH, <ni 2)> daň z příjmu
nm	In formula	<nm q = 3m + p>
np	Part of personal name	Karel <np IV.>
nq	Town quarter	Praha <nq 8>
nr	Ratio	v poměru <nr 2:3>
nw	Flat size	byty <nw 3+1>
n_	Underspecified	autobusová linka <n 158>

Tag	Meaning	Examples
o	Artifact names	
oa	Cultural artifact (book, movie, ...)	Hugovi <oa Bídníci>
oc	Chemical	plyny <oc Ar> a <oc He>
oe	Measure unit	200 <om MHz>
om	Currency unit	1000 <om Kč>, 30 <om $>
op	Product	mikroprocesory <op Intel Pentium>
or	Directive/Norm	ve <or Sbírce zákonů>
o_	Underspecified	čeleď <o Rubiaceae>, <o HIV>
p	Personal names	
pb	Animal name	pes <pb Fík>
pc	Inhabitant name	<pc Afričan>, <pc Pražan>
pd	(Academic) title	<pd Mgr.> J. Pola
pf	First name	<pf David> Uhl, <pf J.> Drda
pm	Second name	Georg <pm Friedrich> Händel
pp	Religion/Myth persons	<pp sv. Jakub>, <pp Prozřetelnost>
ps	Surname	<ps Nováková>, <ps van Dyk>
p_	Underspecified	<p Slované>
q	Quantitative expressions	
qc	Cardinal number	<qc 168> stran, <qc 3> %
qo	Ordinal number	<qo 6.> děkan, přišel jako <qo 3.>
t	Time expressions	
tc	Century	<tc 20.> století
td	Day	<td 1.> října 2005
tf	Feast	<tf Velikonoce>
th	Hour	v <th 17> hodin
tm	Month	1. <tm října> 2005
tn	Minute	<tn 25> minut
tp	Epoch	<tp 70.> léta, od <tp dvacátých> let
ts	Second	<ts 3> sekundy
ty	Year	od roku <ty 1555>

Table 2.1: Czech two-level classification of NEs

2.3 Data

We use the Stanford NER for the recognition of named entities in English, and therefore we do not need any English data.

Tag	Meaning	Example
A	Adress	<A<if KOMO>, <gs Knížecí> <ah 12/173>, <az 709 00> <gu Ostrava-Nová Ves>, tel.: <at 069 6621773>, <at 6621375>, <at 601527588>, fax: <at 069 6621773>>
C	Bibliographical entry	<C<P<pf G.> <ps Lukács>>: <oa<f Die Theorie des Romans>>, <gu Berlin>, <ic<f Verlag <P<pf Paul> <ps Cassirer>>>> <ty 1920>>
P	Name of a person	<P<pd Doc.> <pd MUDr.> <pf Přemysl> <ps Doberský>, <pd DrSc.>>
T	Time	<T<td 21.> <tm června> <ty 2003> <th 20.00>>

Table 2.2: Containers

Tag	Meaning	Example
s	abbreviation	je členem <io<s ODS>>
f	foreign language word	<if<f Deutsche Bank>>
segm	the word is capitalized eg. because of an erroneous segmantation of the text	Revoluční zvrat v pohledu na život <segm Základem> života není...
cap	the word is capitalized for typographical reasons	A jak <cap TO> vysvětlíte?
lower	the word is capitalized when it should not be	ekonomicky ovládnout <lower Střední> <gt Evropu>
upper	the word is not capitalized when it should be	dalším zajímavým <upper britem> je...
?	entity of an unspecified type	<? Asmara> se odmítá stáhnout z území...
!	unannotated sentence	<!> 70: 15 Písnína S. Georga, op.

Table 2.3: Other tags

Regarding the data for training and testing of the Czech NER, we did not have the time, money, or manpower needed to manually annotate a representative corpus of a sufficient size. Therefore, we decided to use the already existing Czech Named Entities Corpus. However, because the Czech Named Entities Corpus has a higher-than-average proportion of NEs, we also use some additional data manually annotated by us to compensate for that.

Czech Named Entities Corpus

The Czech Named Entities Corpus [13] has been used as the basis for the data. For its creation, 6000 sentences were randomly selected from the Czech National Corpus from the result of the query `([word=".*[a-z0-9]"] [word="[A-Z].*"])`. The query was tailored to result in a selection with a density of NEs higher than the average of the corpus to make the manual annotation more effective. The selection was then manually annotated using a two-level NE classification (see section 2.2) and then cleaned; some sentences (for example the ones with wrong segmentation in the original data) were removed. The final size of the corpus is 5866 sentences. [6] Because of the query used for selection, the corpus is not representative with regard to the proportion of NEs in the sentences – every sentence in the Czech Named Entities Corpus has at least one (non-sentence-initial) word that is capitalized, and therefore a high probability of having a named entity. That means there are little to no sentences without any NEs in the Czech Named Entities Corpus, which is not true about the Czech National Corpus.

The Czech Named Entities Corpus comes in several formats [13]:

- **original text** – original manual annotations with comments divided into three parts
- **plain text** – manual annotations without comments merged into one part
- **simple xml** – simple xml format
- **tmt − xml** format from TectoMT with morphological analysis
- **html** – html with highlighted named entities

We have used two files based on the Czech Named Entities Corpus: 1. A file

that was created from the plain text format by stripping the NE annotation and adding automatic morphological tags while keeping the line-alignment with the other files. 2. The *plain text* format with some (slight) additional cleaning. There were a few instances that did not conform to the classification and were corrected:

- a single instance of `<I ...>` as a container for institution was split into two `<ic ...>`'s
- all instances of `<m_ ...>` for a news agency were corrected to `<mn ...>`
- one instance of `<sf ...>` (marking an abbreviation and at the same time – wrongly – a foreign word) was corrected to `<s ...>` (marking an abbreviation)
- a single instance of `<qu Popradu>` was corrected to `<gu Popradu>`
- a single instance of `<gy 1922>` was corrected to `<ty 1922>`
- all instances of `<ti ...>` used for marking time interval were changed to a container `<T ...>` containing two or more *t*-supertype entities.
- all instances of one-letter tags were changed to the correct two-character tags

No sentences were removed during the additional cleaning.

Additional Data

The Czech Named Entities Corpus contains 5866 sentences, each of them with at least one capitalized non-sentence-initial word. In the Czech National Corpus, more specifically the syn2000, there are 7639321 sentences in total, out of which 2393206 sentences have no capitalized non-sentence-initial words, which is slightly more than 1/3 of the corpus. To preserve this proportion in our data, we decided to add about 3000 sentences with no capitalized non-sentence-initial words to the data from the Czech Named Entities Corpus. The additional sentences were randomly selected from the Czech National Corpus (specifically syn2000 to be consistent with the Czech Named Entities Corpus) from the result of the query `<s> [word="[A-ZČĎŇŘŠŤŽ].*"] [word="[^.?!A-Z0-9ČĎŇŘŠŤŽ].*"]+ [word="[.?!]"]</s>`.

The additional sentences were then cleaned. In the first pass, most of the sentences with a sentence-initial capitalized NE were removed from the set. Subsequently, two files were created:

1. an unannotated file with lemmas and morphological tags for each word,
2. a plain text file for following annotation – similarly as with the data from the Czech Named Entities Corpus.

The files were kept line-aligned. The plain text file was then annotated according to the guidelines for the Czech Named Entities Corpus – the NEs in the additional data were mostly of the *t*(ime)-supertype, but there were also some previously overlooked entities of other types.

Datasets

The data from both sources were shuffled together (while preserving the line-alignment) by the `shuffle_pairs.pl` script; the resulting files are `all_anotated.txt.shuffled` and `all_plain.txt.shuffled` (8866 sentences, 196209 words). The resulting file was then divided into three disjoint datasets in roughly 8:1:1 ratio by the `divide_data.sh` script. About 80 % of the data serve for the training (`train.anot.txt` and `train.plain.txt`; 7266 sentences, 161246 words), 10 % as development testing data (`dev.anot.txt` and `dev.plain.txt`; 800 sentences, 17409 words), 10 % as evaluation testing data ((`test.anot.txt` and `test.plain.txt`; 800 sentences, 17554 words). There is 5897 instances of NEs, ie. of type tags (see table 2.1), container tags (see table 2.2) and other tags (see table 2.3. Out of this number, there is 4845 instances of NEs in the training data. Out of the 4845 instances of NEs in the training data, 3840 has a span of one token, and 1005 contains more than one token, as shown in Table 2.4. The list of all tags in the training data sorted by the number of occurences is in table 2.5. 188 of those are the container tags – the list of container tags sorted by the number of occurences is in table 2.6.

Span	Number of occurrences
1	3840
2	619
3	271
4	70
5	16
6	13
7	8
8	3
9	3
10	1
15	1

Table 2.4: Spans in training data

2.4 Features

Our task was a reimplementation of the recognizer, so we didn't focus on the development of new features. We have used more or less the standard features also used in [Kravalová, Žabokrtský] [6]. In addition to those, we added a set of context features checking the lemma of the preceding and the following word. We made lists of the "hint" lemmas (one for the preceding and one for the following lemma) including all lemmas seen before or after named entity of all types with poinwise mutual information higher than 5. Poinwise mutual information was computed on the training dataset.

- *morphological features* – part of speech, gender, case, number
- *orthographic features* – capital letter at the beginning of the word, regular expressions for time, dates, or years
- *lists* of known named entities – boolean features describing whether a word is listed in lists of the most common Czech names, surnames, cities, city districts and streets, countries, common objects, famous institutions, and months
- *lemma* – some lemmas contain shortcuts describing the property of a lemma, e.g. "Prahou" (Prague, 7th case) would lemmatize to "Praha ;G" with mark " ;G" hinting that "Praha" is a geographical name.

Type tag	**Number of occurrences**	**%**	**Type tag**	**Number of occurrences**	**%**
pf	574	11.85 %	or	21	0.43 %
gu	469	9.68 %	tc	19	0.39 %
qc	407	8.40 %	gt	19	0.39 %
ps	387	7.99 %	na	17	0.35 %
s	208	4.29 %	lower	17	0.35 %
ic	194	4.00 %	tp	13	0.27 %
gc	184	3.80 %	pd	13	0.27 %
ty	168	3.47 %	om	13	0.27 %
th	159	3.28 %	oe	11	0.23 %
if	156	3.22 %	gh	10	0.21 %
oa	145	2.99 %	tf	9	0.19 %
P	139	2.87 %	g_	9	0.19 %
ni	125	2.58 %	T	8	0.17 %
io	123	2.54 %	nc	8	0.17 %
td	112	2.31 %	oc	7	0.14 %
f	107	2.21 %	mt	7	0.14 %
p_	95	1.96 %	i_	7	0.14 %
op	92	1.90 %	cr	7	0.14 %
segm	89	1.84 %	pb	6	0.12 %
?	86	1.78 %	cp	6	0.12 %
pc	75	1.55 %	mr	5	0.10 %
cap	64	1.32 %	ti	4	0.08 %
n_	56	1.16 %	gp	4	0.08 %
cn	47	0.97 %	cs	4	0.08 %
qo	42	0.87 %	tn	3	0.06 %
ia	42	0.87 %	nm	2	0.04 %
mn	40	0.83 %	A	2	0.04 %
gr	36	0.74 %	upper	1	0.02 %
tm	33	0.68 %	ts	1	0.02 %
o_	32	0.66 %	sf	1	0.02 %
gl	30	0.62 %	m_	1	0.02 %
pp	24	0.50 %	cb	1	0.02 %
gs	22	0.45 %	C	1	0.02 %
gq	22	0.45 %	az	1	0.02 %

Table 2.5: All tags in the training data sorted by the number of occurrences

Container tag	Number of occurences	%
P	171	90.96
T	12	6.38
A	4	2.13
C	1	0.53

Table 2.6: Container tags in the training data sorted by the number of occurences

- *context features* – similar features as the features mentioned so far for the preceding and consecutive words

Because of our use of the CPAN SVM classifier, we transformed all the features into binary ones. This transformation gives us more than a 300-dimensional feature space. All features are described in more detail in the following sections.

2.4.1 One-word model

The one-word model is used for recognition of the one-word named entities. For training this model we used these features:

- Part of speech, gender, number and case of pre-previous word, previous word and actual word - transformed into binary features
- Detection of improbable part of speech tags for a named entity (1 if POS of word is an adverb, interjection, conjunction, pronoun, verb, preposition, particle or punctuation; 0 elsewhere)
- Features for hints from lemma - checks for values
  /Y S E G K R m H U L j g c y b u w p z o/
- Capital letter at the beginning of the lemma
- More capital letters at the beginning of the lemma
- Matching for the time regular expression:

```
^([01]?[0-9]|2[0-3])[.:][0-5][0-9]([ap]m)?$
```

- Suffix of the lemma is 'ová'
- Matching for the year regular expression:

  `^[12][[:digit:]][[:digit:]][[:digit:]]$`
- Capital letter at the beginning of the word form
- Is the lemma listed in the lists of: months, cities, city districts, streets, first names, surnames, countries, objects, or institutions
- Is the word form listed in the lists of: months, cities, city districts, streets, first names, surnames, countries, objects, or institutions
- Feature for hint from lemma of the previous word - checks for value /Y/
- Is the previous word lemma listed in the list of names
- Is the previous word lemma listed in the list of months
- Is the previous word lemma '/'
- Is the previous word lemma '.'
- Does the previous lemma match month number regular expression:

  `^[1-9]$` or `^1[012]$`
- Feature for hing from lemma of the next word - checks for value /S/
- Is the next word lemma listed in the list of surnames
- Is the next word lemma listed in the list of objects
- Is the next word lemma /
- Is the next word lemma .
- Does the next word lemma match the year number regular expression

  `^[12][[:digit:]][[:digit:]][[:digit:]]$`
- Are previous and next lemmas listed in the lists of hint lemmas with high pointwise mutual information (computed on training data)

2.4.2 Two-word model

The two-word model is used for the recognition of the two-word named entities. We look at the word pairs. For training this model we used the following features:

- Part of speech, gender, number and case of previous word, first and second word in the word pair - transformed into binary features
- Detection of improbable part of speech for named entity (1 if POS of first word in word pair is an adverb, interjection, conjunction, pronoun, verb, preposition, particle or punctuation; 0 elsewhere)
- Detection of an improbable part of speech for a named entity (1 if POS of second word in word pair is adverb, interjection, conjunction, pronoun, verb, preposition, particle or punctuation; 0 elsewhere)
- Is the joint lemma of the word pair listed in the lists of: months, cities, city districts, streets, first names, surnames, countries, objects, institutions?
- Is the lemma of the first word in the word pair listed in the lists of: months, cities, city districts, streets, first names, surnames, countries, objects, institutions?
- Is the lemma of the second word in the word pair listed in the lists of: months, cities, city districts, streets, first names, surnames, countries, objects, institutions?
- Is the word lemma of the second word in the word pair: *. / ( ) % : ,*
- Does the word form of the first word in the word pair match day number regexp `^[1-9]$` or `^[12][[:digit:]]$` or `^3[01]$`
- Does the word form of the first word in the word pair match month number regexp `^[1-9]$` or `^1[012]$`
- Capital letter at the beginning of the lemma of the first word in the word pair
- Capital letter at the beginning of the lemma of the second word in the word pair

- Capital letter at the beginning of the form of the first word in the word pair
- Capital letter at the beginning of the form of the second word in the word pair
- Capital letter at the beginning of the form of the first word in the word pair and form of the second word in the word pair is ”
- Features for hints from lemma of the first word in the word pair - checks for values /Y S E G K R m H U L j g c y b u w p z o/
- Features for hints from lemma of the second word in the word pair - checks for values /Y S E G K R m H U L j g c y b u w p z o/
- Is the previous word lemma ”
- Is the previous word lemma ’/’
- Does the next word lemma match the year number regexp `^[12][[:digit:]][[:digit:]][[:digit:]]$`

2.4.3 Three-word model

The three-word model is used for the recognition of the three-word named entities. We look at the word triplets. For training this model we used the following features:

- Is the joint lemma of the word triplet listed in the lists of: months, cities, city districts, streets, first names, surnames, countries, objects, or institutions
- Is the lemma of the first word in the word triplet listed in the lists of: months, cities, city districts, streets, first names, surnames, countries, objects, or institutions
- Is the lemma of the second word in the word triplet listed in the lists of: months, cities, city districts, streets, first names, surnames, countries, objects, or institutions
- Is the lemma of the third word in the word triplet listed in the lists of: months, cities, city districts, streets, first names, surnames, countries,

objects, or institutions

- Is the word lemma of the second word in the word triplet: '-', 'a', 'v'

2.4.4 Container patterns

Some consecutive named entities (patterns) can be logically grouped together. For example a pattern composed of an academic degree followed by a firstname, followed by a surname ("ing. Jan Novák") is name of a person. For this reason we define containers which group similar entities together.

Recognition of the containers is actually not a regular recognition task within the meaning of named entity classification. We use maximum likelihood estimate to predict containers. The container model keeps information about the frequencies of the individual patterns in the training data. In case where a frequent pattern appears in the recognized text, this pattern is marked with an appropriate container label. Table 2.2 shows the available container types.

2.5 Parameter Tuning

Parameter tuning is needed in order to achieve better results during classification with a Support Vector Machine (SVM) model. For tasks such as classification, a SVM can work in infinite-dimensional space to construct a set of hyperplanes separating the data. It uses the notion of a functional margin to determine the best separation, which is defined as the longest distance to the nearest training data point. Intuitively, a larger functional margin results in lower generalization errors, so the best hyperplane is considered to be the one with the largest functional margin.

The functioning of an SVM depends on its kernel's parameters and a soft margin parameter C. We use a Gaussian radial kernel, which is very commonly used, and has a single parameter γ gamma. The goal of parameter tuning is to find the best combination of values for γ and C, which is commonly done through a grid search with exponentially growing values. It is a time-consuming process, in this case some of the computations took more than four days to complete using a ufal computing cluster. The one-word

model was trained on all combinations of the following sequences of values, exponentiated with base 2:

- γ : -15, -13, -11, ... 1, 3, 5
- C : -5, -3, -1, ... 11, 13, 15

Each combination of γ and C values is checked using a 5-fold cross validation technique. Due to the time-consuming nature of finding the right combination of parameters, the two-word and three-word models were subsequently trained with the winning combination of parameters for the one-word model. This does not have a huge effect on the recognition of named entities because the two- and three word models make a relatively small contribution compared to the one-word model.

Each model was evaluated using the TestSVM.pl script, which takes a model and runs it on test data to compute the following measures:

- $precision = TP/(TP + FP)$
- $recall = TP/(TP + FN)$
- $f\text{-}measure = 2 \times precision \times recall/(precision + recall)$

where TP means true positive (i.e. the number of correct results), FP means false positive (i.e. the number of false results interpreted as correct), and FN means false negative (i.e. the number of correct results interpreted as false). The F-measure measures the total accuracy of the model.

2.6 Evaluation

Table 2.7 shows the F-measure results for the different combinations of parameter values when evaluating the model based on its ability to correctly identify the span of entities. The combination which resulted in the highest F-measure score is $C = 2^{-5}$ and $\gamma = 2^5$ with a value of 76.36. Similarly, table 2.8 shows the F-measure results for the same combinations of values when evaluating the model's ability to identify the entity types. The F-measure shown in bold corresponds to the parameter values which showed the highest F-measure score for entity span. Although there are three parameter combinations which score slightly higher than $C = 2^{-5}$ and $\gamma = 2^5$ for identifying

entity type, we decided that identifying the span is a more basic and therefore more important task. Furthermore, the accuracy gain from sticking to these parameters instead of using the ones corresponding to the best identification of entity types, was quite significant; another good reason to choose the best combination from the first table.

$\downarrow \gamma \mid \mathbf{C} \rightarrow$	2^{-5}	2^{-3}	2^{-1}	2^{1}	2^{3}	2^{5}	2^{7}	2^{9}	2^{11}	2^{13}	2^{15}
2^{-15}	0	0	0	0	35.42	66.54	67.81	71.24	71.53	71.61	71.73
2^{-13}	0	0	0	35.39	66.53	67.83	71.24	71.56	71.63	71.85	72.03
2^{-11}	0	0	35.20	66.49	67.82	71.26	71.60	71.73	72.49	73.33	74.73
2^{-9}	0	34.48	66.37	67.85	71.26	71.67	72.54	73.82	75.22	75.70	75.50
2^{-7}	31.70	65.07	67.54	71.33	72.06	73.66	75.59	75.89	75.79	75.35	75.22
2^{-5}	54.35	67.31	71.45	73.17	75.20	**76.36**	76.22	75.69	75.54	75.50	75.53
2^{-3}	62.60	69.46	73.32	75.66	76.30	75.87	75.78	75.82	75.82	75.82	75.82
2^{-1}	27.37	50.63	64.07	69.92	70.18	70.18	70.18	70.18	70.18	70.18	70.18
2^{1}	17.49	32.53	48.45	59.07	59.07	59.07	59.07	59.07	59.07	59.07	59.07
2^{3}	16.89	32.08	48.70	58.57	59.07	59.07	59.07	59.07	59.07	59.07	58.90
2^{5}	16.89	32.08	48.70	58.90	59.07	58.57	59.07	59.07	59.07	59.07	59.07

Table 2.7: F-measure on span depending on the SVM parameters

$\downarrow \gamma \mid \mathbf{C} \rightarrow$	2^{-5}	2^{-3}	2^{-1}	2^{1}	2^{3}	2^{5}	2^{7}	2^{9}	2^{11}	2^{13}	2^{15}
2^{-15}	0	0	0	0	31.57	51.81	55.04	58.64	59.59	59.66	59.81
2^{-13}	0	0	0	31.55	51.81	55.06	58.63	59.62	59.66	59.84	59.97
2^{-11}	0	0	31.40	51.82	55.05	58.64	59.67	59.74	60.37	61.06	61.92
2^{-9}	0	30.93	51.80	55.14	58.67	59.69	60.20	61.48	62.27	61.67	59.85
2^{-7}	28.80	51.11	54.80	58.74	59.81	61.22	62.60	61.75	59.88	58.65	58.16
2^{-5}	44.70	54.14	58.83	60.78	62.67	**62.28**	60.09	58.70	58.36	58.32	58.33
2^{-3}	51.59	57.28	61.04	62.81	60.95	58.94	58.85	58.86	58.86	58.86	58.86
2^{-1}	26.14	44.81	55.28	57.36	56.85	56.85	56.85	56.85	56.85	56.85	56.85
2^{1}	16.97	30.08	43.20	48.98	48.98	48.98	48.98	48.98	48.98	48.98	48.98
2^{3}	016.40	29.73	43.35	48.18	48.99	48.99	48.99	48.99	48.99	48.99	48.99
2^{5}	16.40	29.73	43.35	48.88	48.99	48.18	48.99	48.99	48.99	48.99	48.99

Table 2.8: F-measure on types depending on the SVM parameters

Using the chosen parameters of $C = 2^{-5}$ and $\gamma = 2^{5}$, the tuned SVM model with a Gaussian radial kernel results in the following values:

- Total precision: 0.723
- Total recall: 0.601

- Total F-Measure: 0.656

The final F-measure is lower than the results found by Kravalová and Žabokrtský (2009) [6], but that was expected. The main problem encountered by the NER implementation from 2009 was that it had been trained on data with a higher than average number of entities. This caused it to overidentify named entities in text. Our data was expanded to include data without any entities, to avoid the problem of identifying the named entities too often in a text. As a result the recall scores drop, and so the F-measure drops as well.

2.7 Testing

Test for the Czech recognizer are placed in its Treex folder. These tests test presence and loading of files with lists of known named entities and all feature functions in `Treex::Tool::NamedEnt::Features::Common` used during the recognition. Recognition scenarios with tricky input are also run.

Chapter 3

Web service

3.1 Overview

The main task of the web service is to receive data from the Chrome extension and "run" it through the Treex scenario. So among other things it has to choose the right language, convert the received data to Treex::Core::Document, serialize the data back and retrieve the information which will be displayed with the named entities.

It's written in Perl on top of the Mojolicious[1] framework.

Terminological note

There is a definition of Web service architecture by W3[2] saying that:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typ-

[1] `http://mojolicio.us/`
[2] `http://www.w3.org/TR/ws-arch/#whatis`

ically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

We choose to use the words "web service" in a less strict manner, meaning application accessible over network. We do not use SOAP nor XML.

3.2 Mojolicious

The following section 3.2 is an excerpt from the Mojolicious documentation to give a brief overview of its function. It might seem like overkill to use this for our task, but the development within this framework appears to be really fast. It comes with its own server implementation and the debug version of the server is quite useful. Overall it contains really useful tools for our application.

Features

- An amazing real-time web framework, allowing you to easily grow single file Mojolicious::Lite prototypes into well-structured web applications.

  Powerful out of the box with RESTful routes, plugins, commands, Perl-ish templates, content negotiation, session management, testing framework, static file server, first class Unicode support and much more for you to discover.

- Very clean, portable and Object Oriented pure-Perl API without any hidden magic and no requirements besides Perl 5.10.1 (although 5.16+ is recommended, and optional CPAN modules will be used to provide advanced functionality if they are installed).

- Full stack HTTP and WebSocket client/server implementation with IPv6, TLS, SNI, IDNA, Comet (long polling), keep-alive, connection pooling, timeout, cookie, multipart, proxy and gzip compression support.

- Built-in non-blocking I/O web server, supporting multiple event loops as well as optional preforking and hot deployment, perfect for embedding.

- Automatic CGI and PSGI detection.
- JSON and HTML/XML parser with CSS selector support.
- Fresh code based upon years of experience developing Catalyst.

3.3 Language selection

Approaches to named entity recognition are usually language dependent (eg. capitalization might be a good hint in Czech, but is quite useless in German). So recognizing the language of a document is vital for correct named entity recognition.

Hence one of the first things the service has to do, is decide what the language of the input is. Currently our recognizer supports only English and Czech. The user of the extension can either state which language he/she wants to use, or allow the service to detect the language of a text automatically.

When no preference is set, the language is identified by Lingua::YALI[3]. It contains pretrained language models for Czech and English which can then be used to simply choose the most probable language given the input text. In our case the input is the text sent to the web service by the user through the browser extension.

3.4 Creating Treex::Core::Document

The next step is converting the input sent by the browser extension to something Treex is able to process. Normally this could be done by removing the markup from the text and applying the scenario. However the situation is made more difficult in this case by the need to return the results to the original tab with minimal damage to the original webpage layout. Therefore we require a more careful approach to convert the input without losing valuable markup information.

Basically, HTML::TokeParser::Simple is used to go through the input. The text tokens (which should correspond to DOM text() nodes) are used to

[3] `http://ufal.mff.cuni.cz/~majlis/yali/`

create "flat" analytical trees. Any other token (eg. a tag) is added as a wild attribute, either 'markupafter' or 'markupbefore' to corresponding nodes of these trees.

Since text tokens can contain relatively large portions of text (such as entire paragraphs), we actually have to run them through a segmenter (that splits it into sentences) and a tokenizer (that splits it into individual words), before we can use them to create anodes. From each segment a bundle is created and all the trees are in a zone denoted by the language. The segmentation and tokenization gets rid of superfluous whitespaces, which usually works fine until you run into preformatted text (PRE tag) where the sequences of whitespaces have an actual meaning.

Finally, after the document is created it's possible to apply scenarios. A scenario is a list of blocks (actions) that should be run on the document. After the segmentation and tokenization are done, the only thing that's left is running a morphological tagger and the actual named entity recognition task. The tagger is not needed for English (and neither is the segmentation/tokenization if we have plaintext), because this is dealt with by the StanfordNER.

3.5 Serialization

We have to turn the documents back into HTML and we also have to provide some way of attaching "user actions" to the recognized entities.

N layer

For each bundle ("sentence") the named entity recognition task added one (possibly empty) n-tree. The nodes in the tree are either named entities (possibly multi-word) and/or containers (eg. fname and lname entities can create a container for a person). Each n-node has references to all the a-nodes it consists of (multiword/containers will reference multiple a-nodes). Similarly each a-node has a back reference to n-nodes (a-node can again reference multiple n-nodes; eg. a container node and one of the container members).

To allow the user actions, each named entity (its serialized form) is wrapped inside a SPAN element. Care must be taken to properly nest these spans in case of containers.

3.6 Data sources

The service also prepares the data to be displayed by the extension. This usually means downloading/parsing from some external resource, or preparing HTML elements as is the case for the Google Maps static API (creating an IMG element). It is assumed that the service will have a somewhat better connection, and better caching capabilities (bigger storage, if it's used better chance of cache hits...)

Google maps API

As a source for our maps we use Static Maps by Google[4]. This way the service just composes the img element that the client should download and display.

Wikipedia abstract

Although Wikimedia provides some kind of API, we chose not to use it. Instead we just use the normalized name of the entity in the URL. As the abstract we take the first paragraph in element with id="mw-content-text". One more thing is done with the abstract and that is turning the relative wiki links into absolute, because the origin usually won't be wiki. The language mutation of the wiki used for our queries is chosen based on the selected language 3.3.

[4] `https://developers.google.com/maps/documentation/staticmaps/`

3.7 API

Our API is quite a simplistic one, based around http protocol. Meaning service side errors produce responses with 5.. status, errors caused by users lead to 4.. status and 2.. status if everything is fine.

All the data are sent in JSON (concrete examples can be seen below).

Endpoints

Currently the service is accessible at `http://ufallab.ms.mff.cuni.cz:12180/`. The JSON file found here provides all the available endpoints the extension/client can use. There should be one endpoint for each language implemented.

Format

Each endpoint is described with two fields: its absolute url and its description. The endpoints are then grouped under "ners". An example of the response follows in 3.1:

Input

For sending the data to the service, you'll need to POST them to one of the available endpoints (see 3.7). The format is yet again JSON; this time with just one field called "selection" which contains the data (text/html snippet) you wish to run the recognizer on. See 3.2.

Output

The response for the request in 3.7 contains the input enriched with spans of the form around the named entities. It also contains id → popup mapping, where

```
1 {"config" :
2  {"ners" :
3   {
4     "auto" :
5      {"url":"http:\/\/ufallab.ms.mff.cuni.cz:12180\/ner\/auto",
6        "description":"Let the service choose the best language
             of the text. Decides between eng and ces."},
7     "czech" :
8      {"url":"http:\/\/ufallab.ms.mff.cuni.cz:12180\/ner\/ces",
9        "description":"Named entity recognizer for Czech. Always
             assume Czech"},
10    "english" :
11     {"url":"http:\/\/ufallab.ms.mff.cuni.cz:12180\/ner\/eng",
12       "description":"Named entity recognizer for English, based
             on StanfordNER. Always assume English"}
13  }
14 }
15 }
```

Listing 3.1: Available endpoints

```
1 {"selection" : "Prague is a nice city."}
2 {"selection" : "<b>Prague</b> is a nice city."}
```

Listing 3.2: Sending the data

```
1 {"html_response":
2         "<span id=\"sysnerv_n_tree-en-s1-n7\" class=\"sysnerv-
              named-entity sysnerv-type-g_ sysnerv-supertype-g\">
              Prague <\/span>is a nice city.",
3
4 "entities_mapping":{
5         "sysnerv_n_tree-en-s1-n7":
6  >>The popup data are omitted<<
7 }
8 }
```

Listing 3.3: Response for "Prague is a nice city"

id is the id of a span and popup is a html snippet to be displayed upon mouseover. See 3.3.

Chapter 4

Browser Extension

The browser extension is the easiest method for users to connect to and utilize the NER implementation. It acts as a wrapper to send the user-selected input to the web service and display the information which is received in return. The extension should have a simple installation process, and be available for analyzing texts while the user browses the web.

4.1 Choice of browser

The first important step in developing the browser extension is the choice of browser. Ideally it needs a large user base to make the extension accessible and an easy-to-use system for extension development. Currently the most popular browsers are Internet Explorer, Mozilla Firefox, Google Chrome, Safari, and Opera [16]. Two of these browsers can be discarded as options almost immediately. The first being Internet Explorer because although it is now possible to build your own extensions for this browser, proper documentation, resources, and support are hard to find. Second the Safari browser can be discarded as an option since developing an extension for it requires the developer to join the Safari Developer Program and obtain a certificate from them. So the final decision rested between the remaining three browsers, all of which have proper support and resources for extension development as well as a reasonably large user base.

Mozilla Firefox has declined slightly in popularity over the last few years, but it still has a relatively large number of users actively developing extensions. Creating an extension requires some preparation by changing configuration settings and creating a separate developer profile, which prevents the settings and changes caused by the development from spilling over into the user's regular profile. Once the browser has been properly set up the extension programming can begin. Firefox extensions require a very specific structure of folders and files with an RDF file containing the most basic meta-information regarding the extension and a manifest file to control the elements of the user interface located outside the content window. Editing the user interface of Firefox requires the use of the special XML User Interface Language (XUL) developed by Mozilla, and the main functionality of the extension can be implemented in JavaScript. To test the extension the developer can create a pointer to the extension folder and it will be loaded every time Firefox is restarted.

Opera is currently the least popular browser out of the three options, but it has good resources and tutorials for extension development. Development starts with an XML configuration file containing basic meta-information and a start file which creates UI elements with HTML. The remaining functionality of the extension can simply be programmed using JavaScript, HTML, CSS, and other common web technologies. During testing the extension can be installed by dragging the configuration file into the Opera browser, which installs it immediately. Although Opera is less popular than Firefox at the moment, the system for developing extensions is less restrictive in structure and only requires the developer to know common web technologies instead of specialized language like XUL. Furthermore the browser does not need any preparation for developers to start working.

Google Chrome has become increasingly popular and also has a large user base actively developing extensions for it. Preparing the browser for extension development requires enabling the Developer Mode on the extensions tool page by simply ticking a checkbox. Developer Mode gives the option to load an unpacked extension which is directly installed and ready for testing. There is no specific folder or file structure for constructing a Chrome extension, but it does require a manifest file to hold the configuration settings and meta-information in the form of a JSON object. The main functionality of an extension needs to be implemented in JavaScript, and the window content can be edited with HTML5 and CSS. Extensions developed for Google

Chrome are also applicable to Chromium, the open source web browser often used on Linux systems, upon which Chrome is based. Due to its popularity and the relative ease of extension development, the Google Chrome browser is the best choice. Preparation of the browser is simple, and the only knowledge required of the developer are common techniques used in web programming.

4.2 User interface

Once the choice has been made to develop the extension for the Google Chrome browser, it is important to make some key decisions regarding the user interface. An entity recognizer needs to be applied to pieces of text, which could either be identified automatically by the extension or selected by the user. Having the extension automatically detect all the text on a webpage would decrease the effort required on the part of the user, but depending on the amount of text sent to the recognizer it might slow down the process significantly and analyze text in which the user has no interest. Having the user select the relevant text they would like analyzed ensures that it is sent to the recognizer and minimizes the waiting time. So in this case the latter option is more suitable. Once the text has been selected, the extension needs to be activated.

The standard options for activating any extension are browser actions, context menu items, or page actions. Page actions are icons which appear inside the address bar of the browser and are not applicable to every page, whereas browser actions are icons which appear in the main Chrome toolbar and are active in every tab, and context menu items can be applied to any page when a certain criteria is met. Since a user should be able to utilize the entity recognizer on any piece of text and most websites do consist of at least some textual components, the latter two options are more suitable. One major advantage of the context menu item is that it can be set to appear only when a piece of text has been selected, whereas a browser action icon is always present and can clutter up the toolbar when a user has many extensions installed. For this reason the context menu item is the most suitable choice for this particular application.

Once the extension is activated, the selected text is sent to the recognizer and entity data is received in return. This data needs to be displayed for the user

in such a way that it does not interfere too much with the original webpage. One option is to have a new window appear with all the information about the recognized entities in one place, and the other option is to have separate popups appear for each entity upon mouse-over. The first causes the least interference original webpage, but the latter allows the user to naturally read the text and receive the extra information exactly when the entity is encountered. For this reason the second option is more appealing. In order to indicate which words do display more information upon mouse-over the extension will make a small adjustment to the original webpage in the form of highlighting those words corresponding to entities. More specifically the words will be highlighted with certain colors based on their entity type. Figure 4.1 shows a screenshot of applying the extension to a web page.

Figure 4.1: Screenshot result of using the extension

In total the extension consists of one background page which is always running and one or more content pages corresponding to the tabs open at any specific point in time. A content page registers when a user makes a selection on the webpage displayed in the corresponding tab, as well as when the user

has pressed the button in the context menu to activate the recognizer. Selections are first split into chunks based on HTML block elements, which will be useful later in the process for decreasing the user's waiting time. After splitting the selection, this information is sent to the background page. This page is responsible for receiving and storing the selections made in different tabs, sending the data to the web service, and relaying the entity data back to the correct content page. For all open tabs the background page stores the most recent selection made by the user. Once the user activates the context menu item in a tab, the background page sends each block of the latest selection corresponding to that tab to the web service via an AJAX request. Upon receiving the selection back from the web service with the entities marked and a mapping of the information retrieved for each entity, the background page passes this back to the content page. The individual blocks are sent back in the order which the background page receives them, which means the smallest block or the one containing the fewest entities is usually replaced first. Another option would be to wait until the whole selection has been analyzed, but splitting the work and replacing the content as soon as the analysis is finished reduces the wait time experienced by the user. Finally, the content page is responsible for replacing the user's original selection on the webpage with the enhanced text created by the web service and placing the additional information in the corresponding popups. Up until the time that the first block of the selection has been replaced the extension will show a loading screen, but this disappears afterwards to allow the user to view what is already there while waiting for the rest to load.

Finally, there are several options for distributing a Google Chrome extension. It can be hosted privately by the developer and made available for download as a .crx file, or it can be submitted to the Chrome Web Store[1]. The latter requires the payment of a small $5 fee before being allowed to publish extensions, but it has a large user base and therefore provides more opportunity for distribution. It also provides a platform for users to rate the extension and easily provide feedback. So for these reasons the Web Store is overall a better option.

The goal of the extension is to provide a simple, accessible method for receiving extra information on unknown entities while reading a text online. The browser for which the extension is implemented is chosen based on its

[1] `https://chrome.google.com/webstore/category/extensions`

popularity and the system it uses for extension development. Google Chrome covers both of these areas very well. Although the extension should provide extra information, it should not interfere with the original webpage too much, which can be achieved through limiting the changes to highlighting the text and having popups display the information. The users can select exactly the text which they want to know more about, and the extension can only be activated when this text has been properly selected. This reduces the clutter of having too many extensions installed while at the same time making the functionality available for every page where text is selected.

It was not possible to create automatic tests for the extension, so it was tested manually. Testing was done on different webpages, for both English and Czech, while having several tabs request information at the same time, and with several people using the extension at the same time. There are a few issues and bugs described in section 6, but the general functionality of the extension was operational in these cases.

Chapter 5

Development

December

- Division of work – project management and data, the recognizer implementation, the web service, the browser extension

January

- **Recognizer** – Research and analysis; baseline Treex module for NER created
- **Web service** – Research about options for setting up web service; inquiries about hosting on university systems
- **Extension** – Research and analysis relating to browser choice and general design of the extension; discussing and defining functionality; very basic extension framework set up and working, including text selection.

February

- **Recognizer** – Scripts for feature extraction, SVM model training and testing; maximum entropy model training and evaluation framework additional data collection and preliminary cleaning

- **Web service** – Setting up basic framework of web service, hosting; basic connection with extension
- **Extension** – Dummy analysis of selection (until recognizer is up and running), and highlighting of entities on webpage; simple connection to web server established; refined extension framework

March

- **Recognizer** – One-word entities classification; lists of classes for SM scripts; modules for feature extraction – common, one-word, two-word, three-word; feature extraction prepared for training; SVM training; scripts for data shuffling; data cleaning
- **Web service** – Connection with NER system set up; added automatic language detection for distinguishing between Czech and English; testing of whole extension-service-ner pipeline
- **Extension** – Proper connection to recognizer via web service established, basic entity popups working; first attempt at extracting additional entity information (geographical only); distinguish between Czech and English texts; options page for setting language preferences

April

- **Recognizer** – Container pattern extraction script; evaluation framework for Sysnerv SVM model; known named entities lists added; SVM tuning; scripts for distributed training; datasets creation; model training
- **Web service** – Parsing of html input and serializing output; handling of multi-word entities; changing response format to more suitable JSON objects; extraction of maps and wiki abstracts
- **Extension** – Progress image shown during analysis; information extraction scripts for other entity types; moved information extraction and some other functionality to web service; small framework improvements (JSON objects)

May

- **Recognizer** – Gamma parametrization; final testing; data tools creation; documentation
- **Web service** – Improving download speed; correcting of the retrieved wiki abstracts and organizing them into a more readable format, addition of basic caching method for speedup; switch to asynchronous downloading; final testing; documentation
- **Extension** – Bug fixes (relating to multiple requests, working with several tabs, preventing selection in popups, etc.); editing of colors and popup display; split up multi-paragraph texts and send to service separately to decrease the waiting time; added images and prepared for distribution from web store; final testing; documentation

Chapter 6

Conclusions and Further Work

The reimplementation of the Czech named entity recognizer was successful. It resulted in a lower F-measure than a previous implementation, but this result was expected and the score is still relatively good. The project has resulted in a working stand-alone application for applying the recognizer, and a webservice and Chrome browser extension system for using the recognizer online. In this section we evaluate each component of the project and list the possible improvements which could be tackled by future projects.

There are several ways in which future projects can improve the implementation of the Czech named entity recognizer. For this project we did not focus on the development of features, but this could be something to look at in future projects. Both developing new features and finding more ideal combinations of the exisiting features could improve the effectiveness of the recognizer. There are ways to measure the amount of influence a feature has on the recognition process, so weeding out the features that don't contribute much and adding in new and better features certainly wouldn't hurt the results. We chose to use a SVM model for this implementation, but it would be worthwhile to implement and test other types of models like Conditional Random Field (CRF) models and Maximum Entropy (ME) models, to see which model is most effective in classifying the named entities. A slightly easier option would be to test different kernels for the already implemented SVM model, and choose the most effective one.

Of course the biggest improvement in most NLP tasks is the gathering of

more data. This would drastically improve the performance of the recognizer, but it is a time-consuming and expensive task. The best scenario would be to add more annotated data, but unannotated data would also create an improvement.

Regarding the web service, the main issue to focus on would be the speed of analysis. Having the analysis of entities and the retrieving of information take less time would largely increase the user-friendliness of the extension. Currently the parsing of the input takes roughly 0.5 seconds, and the downloading of additional information is the biggest culprit, needing anywhere between 0.2 and 2 seconds depending on the size of the input and the number of entities found by the NER. One attempt at speeding up the process has already been made by adding a cache that stores the last 1000 entities returned by the NER along with the information relating to those entities. This drastically improves the speed when the user is particularly interested in one subject and continues analyzing pieces of text relating to what he or she was reading before. One option for further speeding up the process is to improve the cache and allow the web service to store more entities, or alternatively to investigate methods for generating a list of relatively common entities and storing those in a cache. If those options do not yield better results than the current caching system it would be worthwhile to investigate other options.

Currently the web service uses only the Google Maps API and Wikipedia as sources for additonal information about entities. It would be beneficial to add other sources as a backup, because not every entity has a wiki page and sometimes the geographical locations are not found on Google Maps. Also, in rare cases, the abstracts on Wikipedia do not contain information relative to the entity. These are not huge issues since it doesn't happen very often, but it would make the service even more effective if they were solved. Another option would be to make the web service more specific in the information it tries to retrieve for some entity types. So instead of showing only the abstract from Wikipedia, it could extract some hard facts (like the population or area of a geographical entity) to list along side the information already provided now. The possibilities for information to display are endless (photos, facts, quotes, etc.), so there are plenty of options to explore in future work on this project.

Furthermore there are many possible improvements which can be made to

the Chrome extension, in terms of fixing remaining bugs as well as changes in design. Currently the most prominent bugs relate to the small changes the extension makes in the original webpages of the selection. Sometimes the changes we apply cause other parts of the webpage to be deleted or changed when this was not the intention. The structure, style, and implementation of websites vary widely. Often websites are constructed poorly with incorrect or even incomplete HTML and CSS. Getting the extension to apply changes without a hitch to every website it encounters is a nearly impossible task. However the performance can be improved through extensive testing and making adjustments based on the issues the extension encounters with different sites. Another bug which occasionally plagues the extension is the inability of the DOM getSelection method to always recognize when the user makes a selection. Even though this is the most popular method for retrieving user selections there are certain cases where it fails to recognize the selection, usually when larger DOM elements are involved. Selection recognition could be improved by researching other available methods to see whether they could be applied correctly in the extension, and then implementing them together with the DOM method to improve the chance of recognizing all selections containing any amount of text.

Then there are some changes which could make the extension more efficient and user-friendly. In the basic design of the extension the background page could be substituted with an event page. The only difference between background pages and event pages is that the latter is only loaded when needed, therefore saving space and system resources when the extension is not being actively used at all times. Currently the background page does not negatively affect the systems we've tested the extension on; however the advantage could be noticeable for users that have many extensions with background pages installed and it would just be a more efficient design in general. Another possible modification of the extension relating to the user-friendliness would be to have the paragraphs of a selection update in chronological order. Currently the paragraphs are updated according to whichever one the web service is able to analyze first, so the first paragraph to be updated is the one with the fewest named entities. This reduces the waiting time for the user, but it might not be very useful if it is a text which must be read in chronological order for proper comprehension. Forcing the extension to update the paragraphs in chronological order would lengthen the waiting time a little, but makes more sense for the reading order. Another possible addition related

to the issue of updating is to introduce a second progress message. The first progress message appears during the time interval between the user's request for analysis and the first time the web service returns information. It is then hidden to allow the user to take advantage of the information that has already been supplied while waiting for the rest. However to make it clear that the extension is still processing the remaining paragraphs, it might help to have a smaller progress image in the corner of the window until the whole analysis is finished. This image should not interfere with the user's ability to start reading what has been provided, but it should serve as a reminder that the analysis of the other paragraphs is still to come.

Although it is not a critical issue, it would be beneficial to take advantage of Chrome's internationalization tools for separating the English and Czech versions of the extension. At the moment all the errors, the options page, and progress messages are displayed in English even if the user is analyzing a Czech text. These items could be translated and applied to the extension using the internationalization methods so that Czech users can have the entire extension work in their native language if they prefer. The language would be determined automatically based on location when downloading from the Chrome Web Store and this would not interfere with the extension's ability to analyze texts in both languages. All of the additions and changes mentioned so far are most important and should be the main focus in the future. However when these changes have been applied, it could also be a good idea to recreate the extension for other browsers. This requires a large amount of work to convert the structure of the Chrome extension to structures used by other browsers, but the basic idea and functioning of the JavaScript files remains the same.

In conclusion, there are plenty of options for improving the different components of this project. But we have delivered a solid, basis for which there is no similar application currently available, and upon which these improvements can be applied in future projects.

Bibliography

[1] Masayuki Asahara and Yuji Matsumoto. Japanese named entity extraction with redundant morphological analysis. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 8–15. Association for Computational Linguistics, 2003.

[2] Daniel M Bikel, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: a high-performance learning name-finder. In *Proceedings of the fifth conference on Applied natural language processing*, pages 194–201. Association for Computational Linguistics, 1997.

[3] Andrew Borthwick, John Sterling, Eugene Agichtein, and Ralph Grishman. Nyu: Description of the mene named entity system as used in muc-7. In *In Proceedings of the Seventh Message Understanding Conference (MUC-7*. Citeseer, 1998.

[4] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

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

[6] Jana Kravalová and Zdeněk Žabokrtský. Czech named entity corpus and svm-based recognizer. In *Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration*, NEWS '09, pages 194–201, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[7] Alexander K. Seewald Matthew Laird. Algorithm::svm. http://search.cpan.org/~lairdm/Algorithm-SVM-0.13/lib/Algorithm/SVM.pm.

[8] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 188–191. Association for Computational Linguistics, 2003.

[9] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[10] Lisa F Rau. Extracting company names from text. In *Artificial Intelligence Applications, 1991. Proceedings., Seventh IEEE Conference on*, volume 1, pages 29–32. IEEE, 1991.

[11] Satoshi Sekine. Nyu: Description of the japanese ne system used for met-2. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1998.

[12] Treex. http://ufal.mff.cuni.cz/treex. Accessed: 02-06-2013.

[13] Magda Ševčíková, Zdeněk Žabokrtský, and Oldřich Krøuza. Czech named entity corpus 1.0. Data/software, http://ufal.mff.cuni.cz/tectomt/releases/czech_named_entity_corpus_10/. ÚFAL MFF UK, June 2007.

[14] Magda Ševčíková, Zdeněk Žabokrtský, and Oldřich Krůza. Zpracování pojmenovaných entit v českých textech. Technical Report 2007/TR-2007-36, ÚFAL MFF UK, 2007.

[15] Zdeněk Žabokrtský, Martin Popel, and Mareček David. Treex::core. http://search.cpan.org/~tkr/Treex-Core-0.08663/lib/Treex/Core.pm. Accessed: 05-06-2013.

[16] w3schools.com browser statistics. http://www.w3schools.com/browsers/browsers_stats.asp. Accessed: 19-05-2013.