READ

RECOGNITION & ENRICHMENT OF ARCHIVAL DOCUMENTS

D5.1
Language Toolkit and Resources P1

Maximilian Bryan, Soeren Laube, Nathanael Philipp
ASV

Distribution: http://read.transkribus.eu/

READ
H2020 Project 674943

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 674943

Project ref no.	H2020 674943
Project acronym	READ
Project full title	Recognition and Enrichment of Archival Documents
Instrument	H2020-EINFRA-2015-1
Thematic priority	EINFRA-9-2015 - e-Infrastructures for virtual research environments (VRE)
Start date/duration	01 January 2016 / 42 Months

Distribution	Public
Contract. date of delivery	31.12.2016
Actual date of delivery	28.12.2016
Date of last update	15.12.2016
Deliverable number	D5.1
Deliverable title	Language Toolkit and Resources P1
Type	Report
Status & version	In process
Contributing WP(s)	WP5
Responsible beneficiary	ASV
Other contributors	
Internal reviewers	URO, UPVLC
Author(s)	Maximilian Bryan, Soeren Laube, Nathanael Philipp
EC project officer	Martin MAJEK
Keywords	Language Toolkit, Dictionary, XML, Crawling

Contents

1 Executive Summary

Several modules such as HTR, Layout Analysis, Document Understanding, Table and Forms processing, are dependent or will benefit from language resources. Since it is impossible to deal with all European languages in the project, in this task a toolkit will be developed which will enable users (scholars, computer scientists, memory organisations) to access either directly or via web-services large sets of language resources, or to build adequate language resources (lexicons, corpora, etc.) for their own language of interest with a minimum of effort.

In the current state of the deliverable, a highly configurable tokenizer, text extractors, a crawler and dictionary tools have been implemented.

2 Language Toolkit

The prototype of the language toolkit currently consists of three parts: a tokenizer, text extractors and dictionary tools.[1]

2.1 Tokenizer

In its basic form, a text consists of a sequence of characters. These characters form tokens (words, symbols phrases, ...), separated by spaces and punctuation. The tokenizer's job is to split the character sequence into its tokens, whereas specific rules differ between usecases. The developed tokenizer is an easy to configure tool to tokenize text. Due to the different languages that can occur in Transkribus and different requirements from different users, it is highly configurable, so that any user can change it to their needs. Also, when comparing automatic recognized texts on a token basis, it is important to tokenize the texts to be compared in the same way in order to have comparable results. Configurable are:

Normalization of characters is important when dealing with unicode text. In some cases, character like $\ddot{a}$ are symbolized by one single character, but could also be symbolized by an a with additional information of having two dots ontop. To make recognized texts and error rates comparable, it is important to normalize text.

Characters for dehyphenation are used when a token has been hyphenated at the end of the line. The tokenizer looks for the given characters and a token in the next line with a small leading character. If the two assertations are given, the tokenizer creates a single token out of the two tokens. This is important when calculating word error rates because split words are unlikely to be found in a dictionary, whereas the compound version is likely to exist.

[1] `https://github.com/Transkribus/TranskribusLanguageResources`

Characters for delimitation are used to determine where to split the text and create the list of tokens. Nearly always, the space is used to split text. But often commas, dots, exclamation marks und question marks are found directly next to words, so they are used as delimitation signs often as well.

Characters for delimitation that should be kept are sometimes used when punctuation is used to split character, but should be kept as a single token. The two lists - characters for delimitations and the characters that should be kept - usually are not the same since one often wants to split between spaces and punctuation but does not want to keep the spaces but the punctuation as tokens.

2.1.1 Example

Imagine the following two-line example[2]:

> war ich unschuldig! Konnt ich dafür, daß, wäh-
> rend die eigensinnigen Reize ihrer Schwester mir

To tokenize the two lines we want to split them between spaces and punctuation. Also, there is one word which starts in line one and ends in line two, indicated by an hyphenation sign and a beginning small letter in the next line. The punctuation characters should be used for splitting tokens, but should be kept as tokens themselves. Next, the two lines tokenized by our tokenizer are shown[3], whereas every line is represented by a dedicated list:

> ['war', 'ich', 'unschuldig', '!', 'Konnt', 'ich', 'dafür', ',', 'daß', ',', 'während']
> ['die', 'eigensinnigen', 'Reize', 'ihrer', 'Schwester', 'mir']

2.2 Text Extractors

The text extractors are used to extract text from different kind of sources. Sources can be PDFs, different types of XML and web pages. The extracted text can be used to fill dictionaries, which can then help to enhance the HTR accuracy.

2.2.1 PDF

A very common source of text are PDF documents. The text extractor receives a path to the PDF file. It then iterates through the pages of the document and extracts the text from each page.

[2]Taken from *Die Leiden des jungen Werthers.*
[3]Indicated in JSON list format

2.2.2 XML

XML is a technical way to represent text. In contrast to PDF, extra information can be given, like for example which words have been abbreviated and their expansion. When dealing with XML, this extra information can be used to influence the result of the text extraction of the given files. The downside is that there are many different formats of XML. That means that not every format can be supported, but the current state of the project supports popular formats like TEI and PAGEXML.

The following listing contains an examplary snippet from a TEI-XML file:

```
1 [...]
2 reprehenderit
3 <choice><abbr>i</abbr><expan>in</expan></choice>
4 voluptate velit esse cillum dolore eu fugiat nulla pariatur.
5 Excepteur sint occaecat cupidatat non proident, sunt
6 [...]
```

Listing 1: Examplary snippet from TEI-XML with an abbreviation with a given expansion.

In listing 1, there is the abbreviated word *in* which is written with just the letter *i*. When extracting text from that document, the user can decide if the abbreviated or the expanded form should be extracted.

2.2.3 Web crawler

The web crawler is a special kind of text extractor which does not extract text from files but from webpages on the internet. The crawler receives an URL. It then downloads that page, extracts all text and all the links from that page. Next, all the links are being followed recursively, as long as they do not link to an external web page.

The crawler may extract a lot of unwanted text, like menu buttons, legal disclaimers etc., but it still fills a dictionary with words that might help to enhance an HTR system.

2.3 Dictionary

The third part of the toolkit prototype is the already mentioned dictionary. A dictionary holds information about a tokenized text, with the words and their frequency with additional information about expanded abbreviations or different writing styles of a word. Furthermore it provides character tables with their respective frequencies. Additionally some meta information like name, description, language and date of creation are provided. A dictionary can be used by an HTR system to improve the recognition.

For example, for a sentence like *Some say it shouldn't rain.*, a dictionary would have an entry for each word, but the entry for *shouldn't* would have the additional value *should not*, the expansion.

When a dictionary is saved, this can be done in standardized formats, so that others can use them with their existing tools. For that the character tables and entries are stored as ARPA files, the meta data as a properties files and the character tables additionally are save as CSV files with the explicit unicode name and code of each character.

Further functions are provided to calculate the out of vocabulary rate of a tokenized text and a merge function to merge two dictionaries.