Improved Text Extraction from PDF Documents for Large-Scale Natural Language Processing*

Jörg Tiedemann

Department of Linguistics and Philology
Uppsala University, Uppsala, Sweden
jorg.tiedemann@lingfil.uu.se

Abstract. The inability of reliable text extraction from arbitrary documents is often an obstacle for large scale NLP based on resources crawled from the Web. One of the largest problems in the conversion of PDF documents is the detection of the boundaries of common textual units such as paragraphs, sentences and words. PDF is a file format optimized for printing and encapsulates a complete description of the layout of a document including text, fonts, graphics and so on. This paper describes a tool for extracting texts from arbitrary PDF files for the support of large-scale data-driven natural language processing. Our approach combines the benefits of several existing solutions for the conversion of PDF documents to plain text and adds a language-independent post-processing procedure that cleans the output for further linguistic processing. In particular, we use the PDF-rendering libraries pdfXtk, Apache Tika and Poppler in various configurations. From the output of these tools we recover proper boundaries using on-the-fly language models and language-independent extraction heuristics. In our research, we looked especially at publications from the European Union, which constitute a valuable multilingual resource, for example, for training statistical machine translation models. We use our tool for the conversion of a large multilingual database crawled from the EU bookshop with the aim of building parallel corpora. Our experiments show that our conversion software is capable of fixing various common issues leading to cleaner data sets in the end.

Keywords: noisy text processing, text normalization, parallel corpora.

1 Introduction

Data-driven technique dominate modern natural language processing. Much progress has been reported in various fields of NLP due to advances in machine learning and growing data sets. However, the availability of clean data sets is still a serious bottleneck for most languages of the world. It is common practice to crawl the World Wide Web to extend data sets (see, e.g., [7,8]) and to find resources not only for medium and low density languages but also for the

* This research is supported by the Swedish Research Council (Vetenskapsrådet) through the project on Discourse-Oriented Machine Translation (2012-916).

A. Gelbukh (Ed.): CICLing 2014, Part I, LNCS 8403, pp. 102–112, 2014.

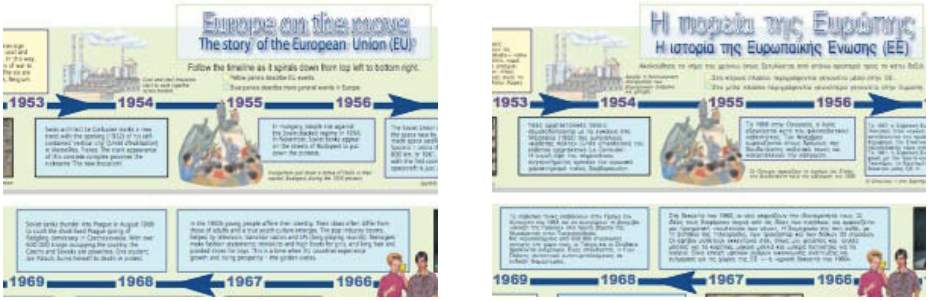


Fig. 1. A screenshot of translated PDF documents (English and Greek) from the EU bookshop with advanced layout

most common languages. The demand for training data is ever growing and domain adaptation problems require additional resources even for the dominating languages on the Internet. Crawling data, however, leads to a serious problem of noise which is unavoidable with the diversity of material available on-line. A large portion of the public documents is not available in clean textual formats that can easily be pushed through common NLP pipelines. In this paper we address the problem of converting PDF documents which is a very challenging task due to the flexibility of that format.

PDF is a file format that is optimized for printing. It encapsulates a complete description of the layout of a document including text, fonts, graphics and other elements. PDF files can be created by a large variety of tools and software products and all of them have their own way of encoding document information in the layout commands defined by the various PDF standards. It is easy to see that the generic conversion of PDF to plain text is a tough problem and probably never solvable with complete satisfaction. However, many documents are available exclusively in that format and, therefore, an extraction of text in the best possible way is an important task for data-driven techniques that rely on this kind of data.

In our work, we are interested in the conversion of documents published by the European Union through the EU bookshop website. Most of the data is completely free and many of the documents are translated into a variety (mostly European) languages, which makes it a valuable resource for many purposes, for example, training machine translation models [5].

One may think that the publications by the EU strictly follow certain standards and style sheets but we have realized that the database is very diverse similar to general web-crawling data. The bookshop includes not only proceedings of the European Parliament but all kinds of genres ranging from treaties, guidelines, and surveys to comics and children's books. Figure 1 shows a typical example of a translated document with advanced layout and structure. Many documents are full of tables, figures and various boxes that make the conversion a tedious task. Design and layout differ a lot and the extraction of text became

the largest challenge when working with the crawled data set. Note that we aim at a batch conversion of massive amounts of data. Manual work is not an option; not even semi-automatic approaches would work on that scale.

After testing a variety of existing solutions we discovered that the final result was still not satisfactory, which lead to our own development of a tool that fixes some of the major issues we found when inspecting the results of preliminary runs. In the following we will first discuss the issues that we address and then present our solutions. In the end we will also describe our data sets collected and converted in our project with statistics and evaluations to emphasize the use of our tool. Note that data sets and tools are freely available through our websites (`http://opus.lingfil.uu.se` and `http://bitbucket.org/tiedemann/pdf2xml`).

2 Converting PDF to XML

Our main goal is to build linguistic resources out of PDF documents. We, therefore, opted for a conversion from PDF directly to a useful XML format that includes proper linguistic boundaries in a standardized and consistent format. However, our conversion tool relies on other software packages as described below which partially produces other types of output (plain text, for example) which we need to handle internally. Several command-line options are available in our practical implementation to control the behavior of our software. In the following, we first discuss the basic tools and libraries we integrate in our package and, then, we introduce the filtering procedures and post-processing features that we have implemented.

2.1 Basic Tools and Parameters

We rely on three public implementations of PDF rendering and conversion software. Firstly, we use the Apache Tika library[1] [6] that comes with various conversion tools not only for PDF documents. Secondly, we integrate the PDF tools provided by the Poppler Developers,[2] a PDF rendering library based on the xpdf 3.0 code base.[3] Finally, we also apply the PDF Extraction Toolkit pdfXtk [2], which is a Java framework built upon PDFBox[4] for document analysis and content extraction.

Apache Tika is a well-known Java-based content analysis toolkit that has a large community within the open-source projects hosted by the Apache Software Foundation and was formerly a sub-project of the information retrieval package Lucene. Apache Tika comes with well-documented API's and command-line tools which makes it easy to integrate the library in other software packages. The Poppler PDF rendering library is a fork of the xpdf toolkit, which is a de-facto

[1] `http://tika.apache.org`
[2] `http://poppler.freedesktop.org`
[3] `http://www.tamirhassan.com/pdfxtk.html`
[4] `http://pdfbox.apache.org`

standard tool in the GNU/Linux world for viewing and converting PDF documents. It includes the text extraction tool *pdftotext*, which we use extensively in our experiments below. Its main advantage is its speed that makes it well suited for large batch processes. The final package, pdfXtk is less known but provides more advanced document analysis techniques based on graph-based wrapping [1], which enables a reliable detection of text boxes in the layout-oriented PDF format. This enables better recognitions of text boundaries based on geometric structure and content attributes such as fonts, styles and font sizes.

All of these tools are able to extract plain text content from arbitrary PDF documents and they all produce reasonable results for most documents. However, we found out that there are still quite a lot of remaining issues that distort the original text and produce garbled results in some cases, which we will discuss in more detail in the following sections.

2.2 Identify Word Boundaries

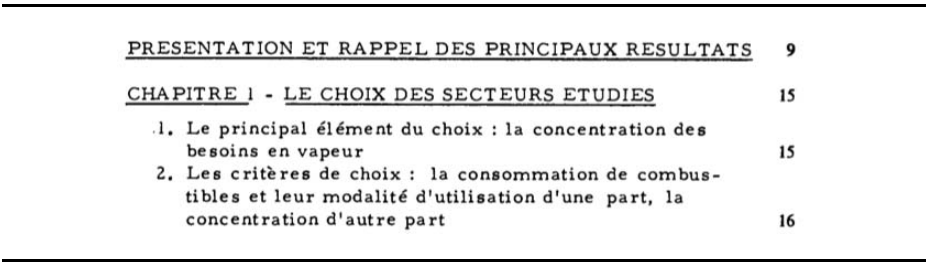
One problem we identified in the output of common conversion tools is the recognition of proper word boundaries.[5] In some cases it is difficult to interpret the spacing between characters and conversion results may look like the sample text in Figure 2. This kind of noise is quite common and distorts the text substantially. For our purposes this kind of output is not acceptable even if the majority of the remaining parts are converted correctly.

As illustrated in Figure 2, the problem of additional spacing is not consistent throughout a document and its appearance is hard to predict. Furthermore, different tools and various modes may result in quite different results as we can see in the lower part of Figure 2, in which the "raw-mode" of *pdftotext* produces a more readable but still not perfect output. Note that the opposite phenomenon also frequently appears; word boundaries are missing and the conversion tools produce concatenated strings of multiple words or entire lines from the document.

Our strategy for handling these problems is based on a data-driven merging and splitting strategy. We run the document through several tools and modes and read the vocabulary from the output of their conversion. Alternatively, word lists can be given to define accepted tokens. Both sources can also be combined. From the data, we then create simple unsmoothed unigram models and use them to process the output of one of the integrated tools to better match the on-the-fly language model. Command-line tools can be used to adjust the conversion modes considered and to select the base tool used for starting the post-processing step. The default settings use *pdftotext* in "raw" and "standard" mode for language modeling and pdfXtk for producing the base conversion. For the latter, Apache Tika can be used as an alternative. Both of them produce XML markup that ensure that we have proper paragraph boundaries which are, otherwise, difficult to detect from raw text output.

[5] We focus here on languages that use spaces to mark word boundaries. For languages without explicit orthographic boundaries, these issues are not apparent.

Original PDF:



PRESENTATION ET RAPPEL DES PRINCIPAUX RESULTATS 9

CHAPITRE 1 - LE CHOIX DES SECTEURS ETUDIES 15

1. Le principal élément du choix : la concentration des besoins en vapeur 15
2. Les critères de choix : la consommation de combustibles et leur modalité d'utilisation d'une part, la concentration d'autre part 16

Converted to text using *pdftotext*:

```
P R E S E N T A T I O N ET R A P P E L DES P R I N C I P A U X R E S U L T A T S

9

C H A P I T R E 1 - L E CHOIX DES S E C T E U R S ETUDIES
1. Le p r i n c i p a l é l é m e n t du choix : la c o n c e n t r a t i o n d e s
b e s o i n s en v a p e u r
2. L e s c r i t è r e s de choix : la c o n s o m m a t i o n de
c o m b u s t i b l e s et l e u r m o d a l i t é d ' u t i l i s a t i o n d'une
p a r t , la concentration d'autre part
```

Converted to text using *pdftotext* in "raw mode":

```
PRESENTATION ET R A P P E L DES PRINCIPAUX RESULTATS 9
CHAPITRE 1 - LE CHOIX DES SECTEURS ETUDIES 15
1. Le principal élément du choix : la concentration des
besoins en vapeur 15
2. Les c r i t è r e s de choix : la consommation de combus-
tibles et leur modalité d'utilisation d'une part, la
concentration d'autre part 16
```

Fig. 2. Problems with word boundary detection in *pdftotext*

Note that we always use Unicode UTF8 to be as flexible as possible and the standard for language modeling is based on lowercased text. Our re-segmentation procedure uses the unigram language model from above together with an efficient inference algorithm based on dynamic programming. We record the longest word in our vocabulary to restrict the history that needs to be considered and run through the string of space separated segments to find possible units that need to be merged. Within this loop we also try de-hyphenation to further improve the results. The highest scoring word sequence according to our language model is then returned as the best output of the post-processing procedure.

Several parameters and heuristics can be used to influence the procedure. First of all, the input sequence can be split into sequences of single character to

force the segmentation to rely entirely on the language model (ignoring the given segmentation provided by the basic conversion tool). This is in general not a good idea and may lead to a decreased conversion quality. However, such a splitting strategy is necessary to handle cases in which words are erroneously concatenated with each other, a problem that frequently appears as well. Therefore, we apply the following heuristics to enable both, splitting and merging: (i) We split strings into single characters if no space character is included in the entire string on one line. (ii) We split tokens that are suspiciously long into single character sequences (i.e. words that are longer than the longest word in the language model). (iii) We split tokens into single character sequences if they contain lower-case letters followed by upper-case letters.[6]

For conversions based on pdfXtk we apply our language-model-based re-segmentation only to those strings that have been split into single characters using the heuristics above. We add another loop that concatenates adjacent words if they exist in the vocabulary. For conversions based on Apache Tika we apply the re-segmentation based on language models on space separated text units. Global splitting into character sequences can be switched on on demand.

2.3 Ligatures

Another problem in the automatic conversion from PDF is the handling of ligatures. The tools we use differ in their capabilities of managing these contracted character sequences. Some of them manage to recognize them correctly and produce single character ligatures as an output. In some cases, ligatures are split and in other cases one (usually the second) character is missing. In our implementation, we normalize existing ligatures using a fixed substitution list. It includes the ligatures for the letter combinations 'IJ', 'ij', 'ff', 'fi', 'fl', 'ffi', 'ffl' and 'st'. Especially pdfXtk has an issue with swallowing some letters if they happen to be part of some specific ligatures. This creates problems especially when applying the merging heuristics described in the previous section. We, therefore, added a test that checks if a character of a known ligature needs to be inserted in order to create a string that is known to the language model. This additional heuristics is quite efficient despite its simplicity and takes care of most of the problems that occur.

2.4 De-hyphenation

Another important issue is hyphenation. In NLP, we usually do not want to have hyphenated words preserved as they appear in formatted texts. We, therefore, add further heuristics to take care of such cases. For this, our line-based post-processing procedure considers two adjacent lines and checks whether the last word of the first line ends with a hyphen. If this is the case then we consider two version in our re-segmentation procedure; one that removes the hyphen and

[6] This heuristic applies to languages that make a distinction between upper-case and lower-case letters as defined by Unicode characters sets.

one that leaves it in place. Our merging heuristics explained earlier then decide whether to concatenate the two strings (last word of the first line and first word of the second line) or not. The software generally prefers the version that allows a concatenation based on having a hyphen or not.

We also apply de-hyphenation heuristics when reading through pre-converted texts when building our language models. In this way, we also obtain words in their de-hyphenated form in our vocabulary and in the language model. It is also possible to test de-hyphenation for all words in the text and evaluate this test based on our on-the-fly vocabulary.

2.5 Paragraph Boundaries

Detecting paragraph boundaries is another important issue that influences subsequent linguistic processing. As we can see in Figure 2, plain text produced by common tools such as *pdftotext* is difficult to work with. Empty lines are simple indicators of new paragraphs. However, many subsequent lines contain paragraph boundaries and without marking them as such, subsequent sentence boundary detection can easily fail. Looking at the example in the figure again, it is not straightforward to identify the start of a new sentence if explicit punctuation and other common linguistic clues are missing. Alternatively, a layout-oriented text format could be chosen. *pdftotext* provides this mode as another possibility and in many cases, this format is much more suited for the recognition of textual units such as lists, headers and paragraphs. However, this mode makes it much harder to handle columns, tables and other formated text that interrupt the normal text flow.

Fortunately, tools such as Apache Tika and pdfXtk address this problem by adding explicit markup for text boxes. Therefore, we use those tools to produce the basic segmentation of texts into coherent segments. In particular, pdfXtk is able to detect very fine-grained boundaries due to its graph-based wrapping algorithm. It is, therefore, our choice for the base conversion. However, pdfXtk tends to over-generate paragraphs and, in this way, it splits many sentences and other coherent elements into pieces. For this reason, we add another simple heuristic to repair common issues and to restore paragraphs based on a simple linguistic rule. Basically, we observe the end of each paragraph and the beginning of the following paragraph in order to make a decision whether to merge them into one or not. In our current implementation we simply merge paragraphs if the first one does not end with a sentence-final punctuation characters and the next one starts with a lower-case letter as defined by the appropriate Unicode character class. This simple rule is very effective and seems to work well in most cases. It can also be switched off on demand.

2.6 Language Detection

Another property that we often require for NLP is that a corpus is homogenous with respect to the language used. PDF documents coming from the European Union, however, are often a mix and may include text written in other languages.

Table 1. The number of documents for the ten largest languages in the EU bookshop collection

language	nr. of doc's
English (en)	37,664
French (fr)	17,260
German (de)	15,585
Italian (it)	9,151
Spanish (es)	7,715
Dutch (nl)	7,687
Danish (da)	7,081
Greek (el)	6,486
Portuguese (pt)	6,380
Finnish (fi)	4,055

This is certainly also the case in other web-crawled data and automatic language identification is a common task that needs to be performed to clean up the data. In our approach, we added an existing language identifier to our software, the Google Compact Language Detector library[7] and its integration into the blacklist classifier for language identification [9]. This feature is optional and can be enabled while converting PDF documents. When switched on, the software runs each paragraph through the language classifier and rejects it if the detected language does not match the given language. This feature is a very useful tool that largely removes unwanted content from our corpora. It actually also helps to remove a lot of garbage that comes from non-text included in many PDF documents or garbled output produced by the PDF rendering libraries. It can be enabled to either remove non-matching text or to just mark each paragraph with the language detected. The latter is useful if subsequent processes need access to language detection information but still require the complete content of the document. This can be the case, for example, for automatic sentence alignment where text removal may cause serious problems.

3 Building a Multilingual EU Bookshop Corpus

In this section, we report our on-going efforts on creating a multilingual parallel corpus from the public documents provided by the EU bookshop. Our collection contains 135,849 PDF documents taken from the official website and many of them are available in various translations. Table 1 lists the ten largest languages represented in the current collection.

We used our tool to convert these documents and we also performed a conversion based on *pdftotext* (in standard mode) as a baseline reference. In our discussion, we focus on four languages: English (en), French (fr), German (de)

[7] `https://code.google.com/p/chromium-compact-language-detector/`

Table 2. Statistics of four disjoint data sets selected from the converted PDF documents. *pdftotext* refers to data sets created using a standard PDF conversion tool and standard linguistic pre-processing (paragraph and sentence boundary detection). *pdf2xml* refers to data created with our PDF conversion tools. The table lists the number of sentences (*sents*), the number of words and the average lengths of sentences in each data set (*w/s*).

	data set A		data set B		data set C		data set D	
lang	pdftotext	pdf2xml	pdftotext	pdf2xml	pdftotext	pdf2xml	pdftotext	pdf2xml
de sents	2.79M	2.93M	0.40M	0.49M	6.90M	7.54M	0.37M	0.39M
words	70.34M	70.46M	10.59M	10.14M	141.76M	134.62M	9.25M	8.92M
w/s	25.224	24.074	26.519	20.762	20.552	17.843	25.334	22.745
en sents	3.65M	3.95M	0.53M	0.65M	33.00M	36.25M	0.60M	0.59M
words	95.96M	95.45M	12.56M	12.01M	621.52M	584.35M	13.71M	13.20M
w/s	26.260	24.189	23.489	18.355	18.833	16.122	23.033	22.224
es sents	1.76M	1.84M	0.25M	0.30M	2.54M	2.74M	0.15M	0.17M
words	54.12M	53.87M	8.52M	8.09M	74.21M	71.07M	6.20M	6.03M
w/s	30.698	29.342	34.073	26.720	29.259	25.954	41.830	36.515
fr sents	1.90M	1.98M	0.46M	0.47M	8.45M	8.83M	0.32M	0.36M
words	57.02M	56.59M	13.36M	12.57M	213.16M	200.80M	12.13M	11.38M
w/s	29.997	28.580	29.270	26.898	25.217	22.737	38.168	31.181

and Spanish (es). We selected four data sets of different sizes based on file name patterns (without implying anything about their contents) to study the differences between the baseline conversion and our improved conversion. In both cases, we run sentence boundary detection after the basic conversion and tokenize the text with the same standard tools. For the *pdftotext* baseline, we also used a simple heuristic rule to improve paragraph detection. Short lines that start with an upper-case letter or a digit are treated as headers which has a great positive effect on subsequent sentence boundary detection. Our length threshold is set to 40 characters. Without this heuristic we would end up with many large text units that would be hard to split later. We also applied the same language detection filter (on the sentence level) in both versions to make the comparison fair.

Let us first look at some statistics from the data selected. Table 2 lists the sentence and token counts for each sub-corpus and each language. Here, we can see some interesting differences between the baseline conversion and our improved conversion. In general, the number of sentences is larger with our tool but the token counts are comparable. This leads to smaller average sentence length which indicates that our character merging and string splitting strategies have a clear effect together with the paragraph boundary detection heuristics.

Certainly, this does not prove that the changes actually improve the data even if manual inspection seems to verify this. Evaluating the general conversion quality is tricky as we do not have any gold standards, and large-scale manual evaluations are too expensive. One possibility is to test the data in a down-stream

Table 3. The test set perplexity of language models trained on 4 disjoint data sets created by a standard PDF conversion tool (*pdftotext*) and our implementation (*pdf2xml*)

lang	tool	data set A	data set B	data set C	data set D
de	pdftotext	467.033	639.193	588.271	553.339
	pdf2xml	464.785	620.899	574.903	530.933
en	pdftotext	314.590	599.654	390.718	556.243
	pdf2xml	312.888	580.897	384.214	541.568
es	pdftotext	256.589	439.685	341.331	415.313
	pdf2xml	256.840	424.248	332.031	401.819
fr	pdftotext	198.377	381.100	233.022	348.049
	pdf2xml	197.642	366.404	226.964	333.529

application. A typical application is the use of such data for language modeling, which is an essential part of many applications. A common metric for showing the appropriateness of a language model given some data is perplexity. Our data is especially interesting for machine translation due to its multilingual contents. Therefore, we selected the news test sets of the SMT evaluation campaign from the annual workshop on machine translation (WMT) from the year 2013. Table 3 lists the test set perplexities measured on these data sets using standard trigram language models trained on our converted PDF documents. We estimated the LM probabilities with KenLM [4] with standard settings (using modified Kneser-Ney smoothing without pruning) and used the KenLM tools [3] for querying the language models.

Concluding from the table, we can see a consistent perplexity reduction on unrelated test data when using our improved PDF conversion as the basis for training language models. The only exception is Spanish on data set A but here, the perplexity is almost identical in both cases. Test set A seems to be the easiest collection as all perplexity scores are very similar anyway. Otherwise, the reduction is quite substantial given that most of the documents are successfully converted with the standard tools as well and only a smaller proportion of the data is actually influenced by the additional post-processing steps.

4 Conclusions

In this paper, we present a new tool for improved text extraction from arbitrary PDF documents. Our approach combines the benefits of several PDF rendering libraries and fixes common problems using several post-processing steps and heuristics. Its main purpose is the creation of large-scale data sets for empirical NLP from noisy and diverse document collections. Our tool manages to improve the detection of word boundaries using on-the-fly language models and efficient re-segmentation procedures. It also normalizes ligatures and removes hyphenations if necessary. The approach does not require external linguistic resources

and is completely open-source and freely available.[8] We used the tool for creating a multilingual corpus of documents published by the European Union. The data sets are also freely available from OPUS[9] [10] and will be useful for cross-lingual applications such as statistical machine translation. In our experiments, we could show that the improved conversion techniques lead to cleaner data sets that reduce the perplexity of unseen test data when measured with a standard n-gram language model trained on the automatically converted documents. In future work, we would like to use the data in machine translation. For this, we need to align all documents pairwise in order to create parallel training data that is applicable for the SMT training pipelines.

References

1. Hassan, T.: Graphwrap: A system for interactive wrapping of pdf documents using graph matching techniques. In: ACM Symposium on Document Engineering, pp. 247–248 (2009)
2. Hassan, T.: Object-level document analysis of PDF files. In: ACM Symposium on Document Engineering, pp. 47–55 (2009)
3. Heafield, K.: Kenlm: Faster and smaller language model queries. In: Proceedings of the Sixth Workshop on Statistical Machine Translation, pp. 187–197. Association for Computational Linguistics, Edinburgh (July 2011), http://www.aclweb.org/anthology/W11-2123
4. Heafield, K., Pouzyrevsky, I., Clark, J.H., Koehn, P.: Scalable modified kneser-ney language model estimation. In: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, vol. 2, pp. 690–696. Association for Computational Linguistics, Sofia (2013), http://www.aclweb.org/anthology/P13-2121
5. Koehn, P.: Statistical Machine Translation. Cambridge University Press (2010)
6. Mattmann, C.A., Zitting, J.L.: Tika in Action. Manning Publications Co. (2011), http://manning.com/mattmann/
7. Resnik, P.: Mining the web for bilingual text. In: Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, pp. 527–534. Association for Computational Linguistics, College Park (June 1999), http://www.aclweb.org/anthology/P99-1068
8. Resnik, P., Smith, N.A.: The Web as a parallel corpus. Computational Linguistics 29(3), 349–380 (2003); special Issue on the Web as Corpus
9. Tiedemann, J., Ljubešić, N.: Efficient discrimination between closely related languages. In: Proceedings of COLING 2012, pp. 2619–2634. The COLING 2012 Organizing Committee, Mumbai, India (2012), http://www.aclweb.org/anthology/C12-1160
10. Tiedemann, J.: Parallel data, tools and interfaces in OPUS. In: Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC 2012), European Language Resources Association (ELRA), Istanbul, Turkey (May 2012)

[8] http://bitbucket.org/tiedemann/pdf2xml
[9] http://opus.lingfil.uu.se