PDF-to-Text Reanalysis for Linguistic Data Mining

Michael Wayne Goodman, Ryan Georgi, Fei Xia
Linguistics Department
University of Washington
goodmami@uw.edu, rgeorgi@uw.edu, fxia@uw.edu

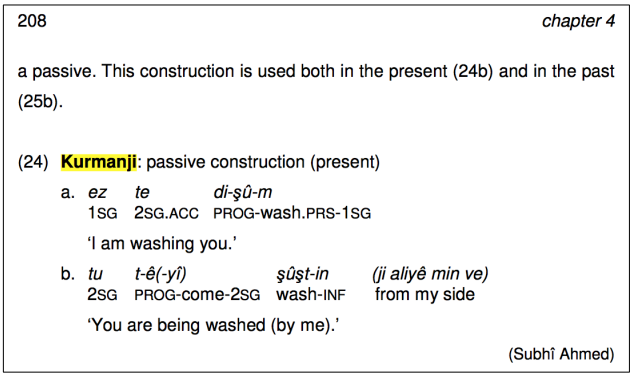
Abstract

Extracting semi-structured text from scientific writing in PDF files is a difficult task that researchers have faced for decades. In the 1990s, this task was largely a computer vision and OCR problem, as PDF files were often the result of scanning printed documents. Today, PDFs have standardized digital typesetting without the need for OCR, but extraction of semi-structured text from these documents remains a nontrivial task. In this paper, we present a system for the reanalysis of glyph-level PDF-extracted text that performs block detection, respacing, and tabular data analysis for the purposes of linguistic data mining. We further present our reanalyzed output format, which attempts to eliminate the extreme verbosity of XML output while leaving important positional information available for downstream processes.

Keywords: Low Resource Languages, Interlinear Glossed Text (IGT), Corpus Creation

1. Introduction

A great deal of linguistic information exists online in the form of academic publications. ODIN, the Online Database of INterlinear text (Lewis and Xia, 2010) was created upon this premise, a resource which makes available language data for approximately 1,500 languages, including linguistic glosses and resource-rich language translations (Lewis et al., 2015). The data targeted by ODIN is Interlinear Glossed Text, or IGT, a semi-structured data format for presentation of linguistic examples, as shown in Figure 1. This data has been shown to have interesting characteristics, making some NLP analysis possible for resource-poor languages (Lewis and Xia, 2008; Georgi et al., 2014; Georgi et al., 2015).


208 *chapter 4*

a passive. This construction is used both in the present (24b) and in the past (25b).

(24) **Kurmanji**: passive construction (present)

a. *ez* *te* *di-şû-m*
1SG 2SG.ACC PROG-wash.PRS-1SG
'I am washing you.'

b. *tu* *t-ê(-yî)* *şûşt-in* *(ji aliyê min ve)*
2SG PROG-come-2SG wash-INF from my side
'You are being washed (by me).'

(Subhî Ahmed)

Figure 1: An IGT instance of Kurmanji (kmr) *in situ* from (van de Visser, 2006).

All the IGT instances in ODIN, including the one in Figure 1, were extracted from linguistic articles that were distributed electronically as Portable Document Format (PDF) documents, a format developed by Adobe Systems for the purposes of standardizing document display and typesetting across platforms. As this format was primarily designed to "communicate visual material between different computer applications and systems" (Warnock, 1991), re-extracting the displayed text for automated systems was not an intended aspect of the format design. In the PDF specification, text is represented as glyphs of a specified font in a "Character Space" coordinate system, embedded within a "Text Space" renderer (Bienz et al., 1997). One of the downsides of this document structure is that the internal structure of a PDF file gives text as glyph-coordinate pairs, and guarantees only the positioning of text rendered on the page. It does not guarantee that the order of the encoded text resembles the order in which it is intended to be read.

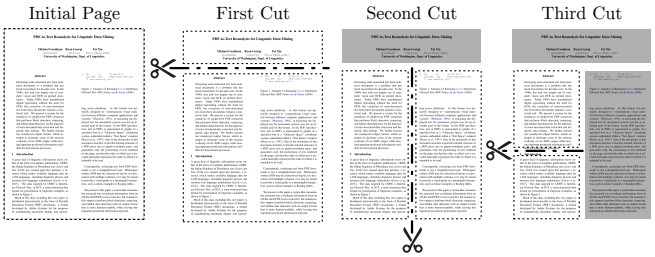


Figure 2: An illustration of the XY-Cut algorithm recursing on a document. The regions created by the current cut (if any) are represented by a dotted border, while regions not under consideration at the current recursion are shaded.

Consequently, extracting text from PDF documents is not a straightforward task. Whitespace within a PDF may be purely a function of layout, as in a document with multiple columns, or it may be meant to provide a cue to meaningful structural deviations in the text, such as inline examples or floating tables.

We present in this paper a system that consumes the extracted text-coordinate information from an off-the-shelf PDF-to-text converter, but reanalyzes this output to perform block detection, respacing, and tabular data analysis. The output format of this system is more human-readable than the verbose xml format produced by the off-the-shelf converters, while making the important positional information available to downstream processes.

2. Previous Work

In the 1980s and 1990s, as access to electronic resources increased, researchers began investigating ways to convert previously printed content into electronic content that could be indexed and searched more easily. A part of the OCR task of transforming pixels into electronic characters was detecting what portions of the page were contiguous blocks. (Nagy and Seth, 1984) introduced the recursive XY-Cut algorithm, with (Nagy et al., 1992) refining the technique for the specific domain of technical articles. For an informative comparison of different algorithms for layout analysis, please see (Shafait et al., 2006).

2.1. The XY-cut algorithm

The XY-cut algorithm works by searching for the largest rectangle of whitespace (or "valley") that runs the length of the current region of the page either vertically or horizontally. This region is then "cut" into two smaller regions, and the algorithm recurses until no more cuts can be found. Rather than cutting only on valleys of complete whitespace, a whitespace threshold can give some tolerance to noise within a valley. Another threshold on the minimum region size to cut can prevent the algorithm from cutting too far. Figure 2 provides an illustration of the first several cuts that the algorithm might make on a sample document.

2.2. Existing PDF-to-text converters

There are several existing PDF-to-text converters, which extract text from PDFs and produce output files with glyph-position information in easily parsable XML formats, such as the open-source PDFMiner (Shinyama, 2016) and the commercial product from PDFLib called Text and Image Extraction Toolkit (TET) (PDFLib GmbH, 2015). The XML output is extremely verbose; for instance, the original PDF document containing Figure 1 is 3MB in size, but the converted XML output of TET, TETML[1] is 58MB uncompressed, 5.5MB with gzip compression. Figure 3 gives an example of the representation of the `PROG-wash-PST-2SG` token from Figure 1 in TETML format.

A third converter we have tested is the pdftotext utility of the Poppler PDF library[2]. A nice property of pdftotext is that it can output text with whitespace to approximate the layout of the page (aka *respacing*). However, the respaced text was often corrupted, with lines of text being split, or with columns becoming misaligned.

Table 1 compares the three converters with respect to features that we find useful when extracting text from scientific documents. All produce markup (XML or HTML) and plain-text formats, where the markup formats contain the coordinates of glyphs and fine-grained blocks (i.e., at word or line-level).

Utility	Respace	Coords	Block	Unicode
pdftotext	✓	✓	✓	basic
PDFMiner		✓	✓	basic
TET		✓	✓	advanced

Table 1: Features of three existing PDF-to-text converters

3. Extraction and Reanalysis

When designing our PDF-to-text system (called Freki[3]), we elected not to reimplement PDF-to-text extraction but instead built on top of off-the-shelf converters that produce the XML format. From the verbose XML format, Freki identifies larger blocks (e.g., a paragraph or an IGT) and performs respacing. It also makes modifications to the XY-cut algorithm.

3.1. Block Detection

The core of our layout analysis is in block detection, which is a two-stage algorithm. In the first stage, we create a 2D array where non-zero values represent the bounding boxes of text tokens in the PDF, then analyze this array for blocks. We aim to create a block for every distinct region of a document, e.g., each paragraph, section header, footnote, figure, etc. The recursive part of the algorithm is similar to the standard XY-cut implementation, but we use three types of thresholds: (1) maximum valley noise; (2) minimum valley width; and (3) minimum non-valley size. Furthermore, all three types of thresholds are defined for both the x and y axes, resulting in six threshold values.

For our task, we set the first threshold to 0 for both axes, as there is little noise to account for in text extracted from generated PDFs. We explore two ways

[1] `https://www.pdflib.com/tet-cookbook/tetml-and-xslt/tetml/`

[2] `https://poppler.freedesktop.org/`

[3] "Freki" is one the wolves that accompanies the Norse god Odin, chosen as a reference to the ODIN project.

```
<Text>PROG-wash</Text>
  <Box llx="238.92" lly="587.60" urx="289.09" ury="597.56">
    <Glyph font="F47" size="8.04" x="238.92" y="587.60" width="5.36" fill="C0">P</Glyph>
    <Glyph font="F47" size="8.04" x="244.32" y="587.60" width="5.80" fill="C0">R</Glyph>
    <Glyph font="F47" size="8.04" x="250.08" y="587.60" width="6.25" fill="C0">O</Glyph>
    <Glyph font="F47" size="8.04" x="256.32" y="587.60" width="6.25" fill="C0">G</Glyph>
    <Glyph font="F47" size="9.96" x="262.56" y="587.60" width="3.31" fill="C0">-</Glyph>
    <Glyph font="F47" size="9.96" x="265.92" y="587.60" width="7.19" fill="C0">w</Glyph>
    ...
  </Box>
```

Figure 3: The TETML output produced by TET for a portion of the token `PROG-wash-PST-2SG` in Fig 1.

```
(24) Kurmanji: passive construction (present)
a. ez te di--m
1SG 2SG.ACC PROG-wash.PRS-1SG
I am washing you.
b. tu t-ł(-y) t-in
2SG PROG-come-2SG wash-INF
You are being washed (by me).
```

Figure 4: The TET plain-text output for the IGT instance in Figure 1.

of choosing the second type of threshold: the first simply uses the average character height in the page for both the x and y axes; the second method analyzes a histogram of valley sizes to select one greater than the median size, which is intended to cut section boundaries but not individual lines. Our experiments show that the first method worked better in general, although the second does better for documents with double-spaced lines. The third type of threshold prevents a cut from resulting in too small a block. We limited vertical cuts so the resulting blocks must be at least 1/6 of the original page width and 1/32 of the original page height, and horizontal cuts so the resulting blocks must be at least 1/6 of the original page width and 1/128 of the original page height.

The second stage of block-detection is line detection. Each block is reanalyzed individually with no minimum valley width for horizontal cuts, which allows each line to be cut. Some PDFs have lines whose bounding boxes are completely abutted, resulting in zero valleys between them. In response, when Freki constructs the 2D array of bounding boxes, it shrinks the boxes vertically by removing 1/5 of its height from the top and bottom.

3.2. Other changes to the XY-cut algorithm

XY-cut, and layout analysis algorithms in general, are often described for analyzing scanned documents. In contrast, we are analyzing the layout of PDFs generated by a typesetting system or word processor, and we ignore non-text elements, so we are not affected by the noise and scanning artifacts that can hinder layout analysis algorithms. Instead, we transform the bounding boxes of encoded glyphs into a 2D array for image analysis. This approach shares some similarities with (Ha et al., 1995), although that work computed bounding boxes for characters from scanned documents, not from PDFs.

XY-cut is a recursive algorithm, so a naïve traversal of the tree of cuts can give an order through the blocks that may correspond well with the intended reading order[4], but block ordering has also been a separate object of research (Meunier, 2005; Sutheebanjard and Premchaiswadi, 2010). We only use the implicit order from the tree traversal, but we record the path to each block, as well as source block coordinates, so that later analysis on block order can be performed. There are also known complex layouts that are not handled by the XY-cut algorithm, but as our focus is on academic articles that do not commonly use such layouts, we choose to ignore these cases for now.

Other work in layout analysis focused on accurately detecting and extracting tabular data (Perez-Arriaga et al., 2016; Oro and Ruffolo, 2009). We also aim to recover data presented tabularly, but rather than being able to accurately predict rows and cells for a table, we aim to maintain visual alignment in tabularly arranged text when converted to a monospaced font, as explained below.

3.3. Respacing and Tabular Data Analysis

Our use case is primarily to enable the automatic processing of text extracted from PDFs, with emphasis on text presented tabularly, such as IGT. Therefore, it is important to maintain the visual alignment of text elements when projecting coordinate-positioned text in a variable-width font to columns in a monospaced plain-text file. If we projected a token's x coordinate to a column by multiplying the coordinate by, e.g., the average character width, words with many narrow characters would occupy proportionally more projected space than words with many wide characters. The effect of this mismatch is that some words would get overwritten by the following words. We could instead multiply the coordinate by the widest character width, or something larger, to prevent the overlap, but this would amplify misalignments across

[4]E.g., for languages that write from left-to-right and top-down, traversing the left/top side of each cut before the the right/bottom cuts is often adequate.

```
doc_id=3667 page=226 block_id=226-6 bbox=... label=tbbtt 6836 6842
line=6836 fonts=F47-10.0,F48-10.0 bbox=...              : (24) Kurmanji: passive construction (present)
line=6837 fonts=F47-10.0,F49-10.0 tabscore=0.25 bbox=...:      a. ez      te      di-şû-m
line=6838 fonts=F47-10.0,F47-8.0 tabscore=1.00 bbox=... :         1SG  2SG.ACC    PROG-wash.PRS-1SG
line=6839 fonts=F47-10.0 bbox=...                       :         'I am washing you.'
```

Figure 5: The Freki output (see Section 4.) for the first IGT instance from Figure 1. The bounding box (`bbox`) information has been truncated to save space.

lines when the original tokens did not have exactly the same x coordinate.[5] We could project to a column, then adjust it to the next available column in cases of overlap, but then tables quickly become misaligned. Instead, we compute the intended column position and compare it to other lines within a block. The proportion of tokens sharing a projected column[6] is a metric we call **tab-score**. Successive lines meeting a threshold for tab-score (which we set to 60%) are marked as belonging to a tabular group. We then apply the project-and-adjust strategy described above, but all tokens within a tabular group sharing a projected column are adjusted to the next available column across all lines in the group. It's possible for non-tabular (i.e., prose) lines to surpass the tab-score threshold, but because we are not explicitly marking the groups as tables and merely adjusting the spacing, the only consequence is extra whitespace between words.

4. Output Format

While XML files produced by TET and PDFMiner offer a great deal of information, as shown in Fig 3, they are both verbose and far less human readable. While disk space may be cheap, reviewability of the output by a human domain expert is necessary to ensure that the resulting content is sensible and useful for consumption by downstream processes.

Compromising between exposing the rich positional information provided by these XML output formats and maintaining human readability, our output format preserves block information and the lines that comprise them, with both block-level and line-level metadata before the text content. Figure 5 shows an example block from the document in which Figure 1 was extracted. The Freki output is more informative than the plain-text output of TET in Fig 4 and more human readable than the XMT output of TET in Fig 3. The information contained in the block and line preambles is explained below.

[5]We've observed a number of documents with ad-hoc tables created, presumably, by the author inserting spaces until the content is visually in the intended spot, but the coordinates will not be exactly the same across rows.

[6]Sometimes things like example numbers inflate the number of unaligned tokens, so we begin counting from the rightmost first token between the two lines.

4.1. Block-Level Information

The first line of the block contains primarily identifying and positional information, including `doc_id` (Source document identifier), `page` (Page number), `block_id` (Unique block identifier), `bbox` (Bounding box of entire block), and `label` (XY-Cut tree path to this block). While the source document and block identifier information is largely for reference, the bounding box, XY-Cut path, and page number are all interesting positional features for downstream processing tasks.

4.2. Line-Level Information

Freki breaks each block into lines of text, which serve as the atomic unit of the output format. Each line is provided with a preamble that contains `line` (line number), `fonts` (list of *font-size* pairs found in the line), `bbox` (bounding box for the line), and `tabscore` (tab-score, see Section 3.3.).

While the line number is not much more than an identifier, the other data exposed are valuable features for the downstream task of identifying the semi-structured, semi-tabular data. For instance, from the bounding box information, one might create a feature for whether a line is offset significantly from the surrounding text, as is common with examples such as the one in Figure 1. The tab-score described in Section 3.3. may help in identifying the word-by-word alignment shown in the first two lines of each example of the IGT instances in Figure 1, as they line up neatly in columns. Finally, the line content following the preamble is additionally respaced to be natural for human readability.

5. Conclusion

Extracting semi-structured text from PDF files is not trivial. While there are existing open-source and commercial PDF-to-text converters, the output is either verbose and not human-readable or lacking important information (such as indention) that can be useful for downstream processes. Our software builds on top of existing converters and focuses on identifying blocks and respacing, useful for targeting IGT. The package, including a web interface, is freely available to the public at `github.com/xigt/freki`.

6. Bibliographical References

Bienz, T., Cohn, R., and Meehan, J. R., (1997). *Portable Document Format Reference Manual*, August. http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf.

Georgi, R., Lewis, W. D., and Xia, F. (2014). Capturing divergence in dependency trees to improve syntactic projection. *Language Resources and Evaluation*, 48(4):709–739, October. http://doi.org/10.1007/s10579-014-9273-4.

Georgi, R., Xia, F., and Lewis, W. D. (2015). Enriching interlinear text using automatically constructed annotators. In *Proceedings of the 9th SIGHUM Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH-2015), in conjunction with ACL 2015*, Beijing, China, July. http://www.aclweb.org/anthology/W15-3709.

Ha, J., Haralick, R. M., and Phillips, I. T. (1995). Recursive xy cut using bounding boxes of connected components. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 2, pages 952–955. IEEE.

Lewis, W. D. and Xia, F. (2008). Automatically identifying computationally relevant typological features. In *Proceedings of the Third International Joint Conference on Natural Language Processing*, Hyderabad, India, January. http://www.aclweb.org/anthology/I/I08/I08-2093.pdf.

Lewis, W. D. and Xia, F. (2010). Developing ODIN: A multilingual repository of annotated language data for hundreds of the world's languages. *Literary and Linguistic Computing*, 25(3):303–319.

Lewis, W. D., Xia, F., Georgi, R., and Goodman, M. (2015). ODIN corpus download. xigt.org/odin.

Meunier, J.-L. (2005). Optimized XY-cut for determining a page reading order. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 347–351 Vol. 1. IEEE, September.

Nagy, G. and Seth, S. (1984). Hierarchical representation of optically scanned documents. In *Proceedings of the 7th International Conference on Pattern Recognition (ICPR)*, pages 347–349, January.

Nagy, G., Seth, S., and Viswanathan, M. (1992). A Prototype Document Image Analysis System for Technical Journals . *Computer*, 25(7):10–22, July.

Oro, E. and Ruffolo, M. (2009). PDF-TREX: An approach for recognizing and extracting tables from PDF documents. In *ICDAR*.

PDFLib GmbH. (2015). PDFLib TET 5 – text and image extraction toolkit. pdflib.com/products/tet/.

Perez-Arriaga, M. O., Estrada, T., and Abad-Mota, S. (2016). TAO: System for table detection and extraction from PDF documents. In *FLAIRS Conference*.

Shafait, F., Keysers, D., and Breuel, T. M. (2006). Performance comparison of six algorithms for page segmentation. In *Document Analysis Systems*.

Shinyama, Y. (2016). pdfminer, December. github.com/euske/pdfminer.

Sutheebanjard, P. and Premchaiswadi, W. (2010). A modified recursive X-Y cut algorithm for solving block ordering problems. *2010 2nd International Conference on Computer Engineering and Technology*, pages V3–307 – V3–311, May.

van de Visser, M. (2006). *The Marked Status of Ergativity*. Landelijke Onderzoekschool Taalwetenschap. http://www.lotpublications.nl/Documents/141_fulltext.pdf.

Warnock, J. (1991). The Camelot Project . blogs.adobe.com/acrobat/files/2013/09/Camelot.pdf.