USAGE AND EVALUATION OF PDF TEXT EXTRACTION METHODS

Erica Sim
Department of Linguistics
University of Washington
Seattle, WA 98105
ejsim@uw.edu

June 14, 2019

ABSTRACT

There are many different tools to extract text from PDFs that are available today, and it can be difficult to know which is the best one to use. This paper describes how many tools can be used simultaneously to extract text from multiple PDFs in batches using a script on patas, the Linguistics department server at University of Washington.

1 Introduction

Extracting text from a PDF is a difficult task that has been tackled many times before, with varying results. The purpose of this paper and the accompanying scripts is to be able to quickly use and compare plain text output from multiple different PDF text extraction tools. There are many reasons why it is useful to be able to easily switch tools on the fly, not to mention why so many text extraction tools exist in the first place. These stem from the inherent nature of PDF as a page description language, meant for printing rather than interacting with text.

In this paper, I will explore the background of PDF as a format and why the task of text extraction is such a difficult one (section 2). Then, I will discuss work that has been done on the patas system to provide usage of a suite of tools collected by Bast and Korzen [4] for their 2017 evaluation of text extraction methods (see 3.3, as well as 3.2 for in-depth information on the tools available). Finally, in section 4, I will discuss some evaluation and comparisons between these tools.

2 Background

2.1 About PDFs

In the years since its introduction in 1993, the Portable Document Format has become a sort of “de facto standard" for the format of electronic documents. [1] There are several reasons for this, which I will discuss in section 2.1.1. I will discuss how the way that PDF encodes and displays a document leads to these benefits in 2.1.1. However, this also causes multiple drawbacks to PDF as a format, as I will discuss in section 2.1.3.

2.1.1 Benefits of PDF

The benefits of PDF as a format are many. [1] states that the intended goal of the format is to “enable users to exchange and view electronic documents easily and reliably, independently of the environment in which they were created." In this, it definitely succeeds. Hassan and Baumgartner state that “any document can be converted to PDF as easily as sending it to the printer, with the confidence that the formatting and layout will be preserved when it is viewed or printed across different computing platforms." [11] This is because a PDF file is optimized for printing and includes all needed information about the layout of the document and all included elements [23].

Many of the biggest benefits to PDF format are the improvements over other solutions of the time. In their 1995 paper “Document Analysis of PDF Files," Lovegrove and Baumgartner are impressed with the “extra semantic clues" afforded when using PDF over working with a raw bitmap. [19] As one example of this, PDF encodes text and images differently, which is incredibly useful for document understanding. The fact that PDF does not require text to be in reading order was considered a feature.

Finally, the fact that a standard exists at all, and can be read (and now written) with free software, are potentially the two biggest benefits provided by PDF.

2.1.2 How the format works

The way that PDF format attains these benefits is by describing data on a page in a unique way (hence it is called a page description language). The objects in the file, be they text or graphics, are saved in a content stream, and then “painted" onto the page when rendered by the PDF [1]. This makes sure that the page will always look the same visually, no matter the software used to display it.

Part of the flexibility of the format is due to the fact that the order the objects appear in the file has no significance to the finished result. Each object has an assigned exact spot where it is located (or painted) on the page, without necessarily any reference to any other object [1]. This is great for images and graphics, but poses an interesting problem when it comes to text. When text is saved as an object, it is not by lines or even words. This text object can be just one character (or rather, glyph; see 2.2.1 for how they relate). Since each object is individually processed and the order is not preserved, there is no easy way to extract the text. To complicate things further, since PDF is such a universal standard, there are a large number of various software and tools that can create them– and each of them can encode the information in the PDF in very different ways [23].

2.1.3 Drawbacks to PDF

The same flexibility of PDF that lead to the benefits in 2.1.1 also causes significant drawbacks to PDF as a format. Since the “objects" can consist of a single character, or a fragment of a word, or even a combination, there is no information encoded about the layout or larger blocks of text. All of this information that we as humans get is from the whitespace [5], which is not directly encoded. One of the biggest hurdles in this area is maintaining reading order in columns of text.

Rendering characters is another major problem for PDFs. The text objects are actually one or more “glyphs," which represent different characters. The font is what matches these glyph shapes to the characters displayed [1]. These fonts can be saved within the PDF, but often they are not. This is fine with very standard fonts, but can cause a huge amount of problems for others. I will further discuss problems due to font encoding in section 2.2.1 below.

PDFs can also pose some major accessibility issues. While Adobe claims accessibility to disabled persons as one of the benefits of the PDF format [1], this is definitively not the case. Çakir examines the case of an important legal document from the government of Germany. This document was created by a publishing company in 2004; in 2012 it was slightly altered and again converted into a PDF document. Although the only difference was the version of Adobe used, there were hundreds of differences in the resulting document, some of which rendered the document unreadable by screen readers. [9] This is a perfect example of the accessibility problem with PDFs– if the text cannot be accurately extracted, this will result in blind users receiving inaccurate or confusing information, or possibly being unable to access the document at all.

Adobe also claims that the PDF format ensures “preservation of document fidelity." [1] While this is certainly the case when measured against other tools and formats of the time (that is, the early 1990s), due to the above problems, this cannot always be assured. What is more, there is often no error message or other way to tell if there has been alterations to or omissions of any characters or whitespace without a human having to check the result carefully.

2.2 Difficulties in text extraction

Many of the problems discussed in 2.1.3 affect the ease and accuracy of text extraction from PDFs. Of course, if a character did not even make it in to the PDF from the original source document, it will not make it into the extracted text. But even when the PDF is visually fine, the way fonts and glyphs are saved in the file can prevent text extraction. Some PDF generators use custom font encodings in order to make the resulting file more compact; this can cause the extracted text to become “gibberish" [7]. Additionally, spacing between words and characters can add more problems, as this can cause any word segmentation algorithms used to make mistakes [5].

As mentioned in section 2.1.3, columns can be one of the biggest culprits in errors, particularly reading order errors. But tables can be even more difficult, as they are inherently column based. They also break up the flow of text, making paragraph boundary detection more difficult [5].

Finally, one of the major problems in text extraction that may be particularly troublesome to computational linguists is the inability to handle mathematical formulæ. None of the tools examined here even attempt to do so. Not only are the characters challenging to deal with (see next section), but they are often drawn with vector operations and so the position vertically can vary (and can obviously change an equation a great deal!) [5]. As most of these tools are intended to be used with scientific papers, leaving out formulæ entirely is not a good solution– but accuracy is also of tantamount importance.

2.2.1 Fonts, glyphs, and characters

As mentioned above in 2.1.3, many problems with PDFs (and extracting text from them) is due to the way that characters and fonts are used and stored (or not) in the file. As mentioned in section 2.1.2, characters are not saved directly, but instead text objects are saved in the content string as glyphs. A glyph is a specific rendering of a character; a font defines the glyphs for each character– this distinction is actually a central tenant of Unicode and not unique to the PDF format [20]. However, the relationship between character and glyph is not 1 to 1. Some characters actually have two (or potentially more) corresponding glyphs [5], such as characters in Arabic or Hebrew scripts, where a character has different glyphs depending on the position within the word. And in the case of ligatures, for example, one glyph actually represents two characters.

There are four main types of fonts used in PDFs. Type 1 and TrueType fonts are the most common and generally pose no problem, particularly the former, which is a set of Adobe's own fonts. Type 0 fonts are also called composite fonts, as they reference multiple other fonts. They are commonly used with non-Latin scripts and generally do not cause many errors [5]. Problems occur much more frequently with Type 3 fonts. These fonts can be generated by the program generating the PDF, and use any and all of the available graphical operators. They don't necessarily (or usually) contain any meta-information about glyphs or the mapping of them to Unicode characters [5].

This Unicode mapping from glyph to character is an incredibly difficult problem and may be considered the very foundation of PDF text extraction [11]. Some of the many issues that can lead to problems include those from diacritics, homoglyphs, legacy compatibility, and the related issue of canonical equivalence. These can lead to unexpected results, particularly by linguists who are often working with multiple languages, potentially with multiple scripts, as well as characters from the International Phonetic Alphabet.

One major problem is homoglyphs, or different characters that are converted into identical (or nearly so) glyphs. For example, the Latin character <A> appears identical to the Cyrillic character <A>. This is extremely widespread in IPA characters; an example there is that the alveolar click <!> actually has a different Unicode code point (and is therefore a different character) from the exclamation point <!> despite looking identical [20]. Different fonts can exacerbate this problem.

A similar, though distinct, phenomenon is canonical equivalence. This is when a character can exist either as one code point or as a combination of base character and a letter modifier [20]. This is mainly present for legacy compatibility (which is ultimately the root of many issues with Unicode encoding). Canonical equivalence differs slightly from regular homoglyphs; they do not merely look identical, but should be treated exactly the same by any program. One example is the character <ñ>. This can (and should) be represented as a single precomposed character with code point U+00F1. However, the sequence of <n> (U+006E) and the combining tilde (U+0303) can also be used. This is intended for legacy compatibility, but still occurs in text extraction [11].

Diacritics are therefore a major source of problems for text extraction. Even as common of a character as <ü> is often converted into two separate glyphs, one for the <u> and another for the umlaut diacritic [11]. The use of multiple diacritics can lead to even more problems, especially if the diacritics are supposed to be placed above or below each other (as is common in IPA transcription) [20].

The level, as well as number, of Unicode blocks used increases the likelihood of errors generated; characters from the Basic Latin block almost never cause any errors [11]. This is a major problem for linguists, as while there is a block named "IPA extensions," however, it does not include characters that are already present in previously assigned blocks. This leads to the full set of IPA characters being spread out upon 12 different blocks [20]. Non-phoneticians are not exempt from these issues, however– even in Spanish, two additional blocks are needed beyond Basic Latin [11].

2.3 Problems and methods in tool evaluation

There is no real agreed-upon objective measure of accuracy of a given text extracted by a given tool. As such, it can be difficult to evaluate the performance of these tools. One could manually inspect the documents, of course, but this doesn't scale to long documents, never mind hundreds of them. Even if this were reasonable, it still leaves the problem of what to look for. Should you just count errors? Are some errors more important than others? What about the differences between the goals and features of these tools? As there are so many, there needs to be a comprehensive evaluation and comparison of the various tools and packages available for text extraction.

Researchers have attempted to solve this problem in various ways. Berg uses the most straight forward approach: In order to assess and compare different text extraction tools, he compiled a list of possible defects that may occur for each document when the text was extracted [5]. The list was then compared manually to the actual output of the tools. This was possible because there were only 15 documents in the set considered, and again due to the human labor necessary for each file, could not be easily scaled. Additionally, he ran into the problem of comparing tools that had different goals and feature sets.

Tiedemann uses perplexity as a measure of evaluation; if a tool creates output with reduced perplexity, it is rated higher [23]. This is a measurement that could actually be scaled; he uses a corpus of over 135,000 documents. However, he only compares his tool (pdf2xml) to one other (pdftotext); this still does not give us our needed comprehensive comparison of tools.

Clearly, comparison would be made easier by utilizing clear and independent criteria to judge quality. Bast and Korzen give four major criteria that are important for a good text extraction: paragraph boundaries, distinction of body text vs. other text, reading order, and word boundaries [4]. From this they chose three points of comparison that can objectively measure the accuracy of an extracted text, by measuring differences between the text and their groundtruth file. These are newline differences, paragraph differences, and word differences. There can be either spurious or missing newlines, paragraphs, and words, as well as rearranged paragraphs and misspelled words. They also noted how many files caused errors and how long it took the tool to process a file. This method of evaluation can be performed on multiple tools in one run.

3 Methodology

3.1 Data set used

The data used was provided by Bast & Korzen; this ensured the files would work with their scripts and allowed better comparison to their results. Their files were scientific articles taken from arXiv.org [25] ranging from 1991 to 2016. They used over 12,000 files in their benchmark and evaluation (about 1% of available articles) [4]. All of these were processed on patas using all available and working tools (see 4.2) and the resulting texts extracted by each tool can be found under `/projects/pdf_eval/code/pdf-text-extraction-benchmark/evaluation/output/data`.

3.2 Tools used

Bast and Korzen examined 14 different text extraction tools in their paper. One of these, pdfx, is intended for use via their website interface, and so is not covered further in this paper. The tools evaluated are:

pdftotext

Pdftotext [21] is one of the most common tools on the Internet used to extract text from PDFs, likely because it has been around since 1996. It is one of many tools that are included with the Xpdf toolkit. It works quickly but does not attempt to identify paragraph boundaries, separate main text, etc. Additionally, it extracts characters with diacritics as two separate characters and doesn't merge hyphenated words. [4]

pdftohtml

Pdftohtml [16] converts the PDF to HTML (or XML) by breaking it down into lines. It is also based on the Xpdf package and so carries the same drawbacks as pdftotext.

pdftoxml

pdftoxml [10] is a tool created by Xerox research something (cite), also using the Xpdf libraries. Confusingly, it is often styled as pdf2xml, which is identical to another tool listed below. [4]

PdfBox

PdfBox [2] is a PDF library from Apache, and gives plain text output. It can handle ligatures and characters with diacritics, but not hyphenation. It does not attempt to identify section or paragraph boundaries. [4]

pdf2xml

This tool [24] uses Apache Tika (which in turn uses PdfBox) and pdftotext. It combines the results of both tools to "improve the detection of word boundaries using on-the-fly language models and efficient re-segmentation procedures." Amazingly, it also utilizes (optional) language detection, by paragraph. [23]

ParsCit

ParsCit [14] uses third party tools and then processes the results to extract text and parse reference strings. Here, it utilizes pdftotext. [4] Its output is in XML format.

LA-PdfText

LA-PdfText [8] utilizes user-defined rules (created for different layouts) to find LTBs and extract text. The default rules only are used here. As such, it can only handle ligatures and not hyphenation or characters with diacritics. [4]

PdfMiner

PdfMiner [22] analyzes the structure of the document and can convert it to XML or HTML as well as plain text. It does not handle ligatures or diacritics well. [4]

pdfXtk

pdfXtk [13] uses PdfBox under the hood and converts the PDF into XML or HTML. It extracts text (as well as graphical content) and converts it to "objects" which are grouped into higher-level structures. It does not merge hyphenated words, and it converts diacritics into a separate character. [12]

pdf-extract

pdf-extract [26] converts PDF documents to XML. It does not identify paragraphs or sections at all. It does not handle diacritics or hyphenation, but it does correctly parse ligatures. [4]

PDFExtract

PDFExtract [6] converts PDFs to XML. It uses geometric layout analysis and logical layout analysis algorithms originally designed for OCR to segment pages into sections, such as the title, abstract, and headings. The only thing it really struggles to handle are tables and formulae. [5]

Grobid

Grobid [18] was created to extract bibliographic information from scholarly articles (its name comes from GeneRation Of BIbliographic Data). [17] It uses Conditional Random Fields to break down a document into various LTBs for XML output. It also handles ligatures, diacritics, and hyphenated words. [4] It is incredibly powerful, but also very slow.

Icecite

Icecite [15] is Bast and Korzen's research paper management system, which includes PDF text extraction [3]. It can have plain text, XML, or JSON output. As expected, it meets all of Bast and Korzen's judgment criteria, including handling diacritics and hyphenation.

3.3 Running the tools on patas

I was eventually able to get 7 of the 13 tools running on the patas server: pdftotext, pdftohtml, PdfBox, PdfMiner, pdfXtk, PDFExtract, and Icecite. These can all be used to extract text from batches of PDFs. There is a shell script intended to help others run these in the future. It can be found at `/projects/pdf_eval/code/pdf-text-extraction-benchmark`.

This a helper script for extractor.py (located in `pdf-text-extraction-benchmark/evaluation/bin`), which in turn can use a multitude of tools to extract text from PDFs. The extractor was originally meant to take large batches of PDFs and extract text using multiple different tools; each tool would then have its own output folder full of plain text files. The intent was then to compare and evaluate the results from these different tools by comparing them against a benchmark.

run_extr.sh can be used in the same way, but the standard use would be to select a text extraction tool to run on a batch of PDFs. (This batch can be as large as you want, but you'll need to use condor if you want to run it on more than 100 or so files, especially if you're using slower or multiple tools.) You can select multiple tools if you are unsure which will provide you with the output that you want. You can find examples of what the output from various tools looks like under `evaluation/output/data`.

It can be called as such:

```
$ run_extr.sh path/to/pdfs path/to/output tool
```

The available options for tools are: pdftotext pdftohtml pdftoxml pdfbox pdf2xml parscit lapdftext pdfminer pdfXtk pdfextract PDFExtract grobid icecite. See next section for more details. You can also type `run_extr.sh-h` for this list of tools (and the spelling/capitalization used for each).

Multiple tools can be used at once. They need to be added as a string, with the names of tools separated by spaces, as such:

```
$ run_extr.sh path/to/pdfs path/to/output "tool1␣tool2␣tool3"
```

The input will need to be a folder of folders of PDFs.

After running the script, in your output folder you will find a folder for each tool you selected. In this folder you will find at least one plain text file for each PDF in your input folder, ending in .final.txt. Some tools may create additional files, such as .raw.txt files. These are often the result of tools that output in XML or HTML; the program tool_extractor.py converts this raw output into text format so that the tools can be accurately judged against each other and the groundtruth benchmark. The .final.txt files are the ones you want to use if you want plain text.

4 Evaluation results and analysis

4.1 Previous results

In Bast and Korzen's results of all 14 aforementioned tools, PdfBox came out on top, with the best score in four of the criteria (lack of spurious new lines and paragraphs, an almost complete lack of out of order paragraphs, and fewest missing words [4]. Additionally, it only failed to process two files, and was the fifth fastest. Pdf-extract was the worst, with the worst or second-worst score in six separate categories: spurious new lines, spurious paragraphs, reordered paragraphs, and spurious, missing, and misspelled words. Pdftohtml handled the most amount of files, with zero failing. On the opposite side, pdf2xml failed on 1,444 files. PDFExtract was the slowest, with Grobid a close second. Pdftotext was the fastest, only taking .3 seconds on average to process a file. Pdftohtml was close, with .7 seconds.

A table of Bast and Korzen's full results can be found in Appendix A.

4.2 Work on Patas

While I was able to get text output from all 12,000+ files for the above mentioned tools on patas, I was unable to evaluate them, as I could not get the evaluator.py script to run without crashing. Getting this script running and getting results from text extracted on patas is the next step. This will allow evaluation of the results and a comparison of output from different tools. (See section 4.2 for more on further work needed on evaluation.)

4.3 In other environments

I also ran the extractor and evaluator scripts on a separate Linux box, which was set up with a fresh install of Ubuntu v16.04 and miniconda3 v4.6. Like on patas, seven tools worked correctly, although they were not all the same tools. On my machine, I was able to get pdftoxml, PdfBox, LA-PdfText, pdfXtk, pdf-extract, Grobid, and Icecite to successfully extract text. Unfortunately, due to the limited power of a virtual machine on a laptop, I was not able to process the full 12,000+ files in the data set. Instead, five folders of articles were chosen at random (using Python's `random.choice()`

method). This gave me a data set of 197 articles, which was much more manageable. However, this means that my results are not as directly comparable to Bast and Korzen's as they would be with the full set.

Figure 1 shows the results of six of these tools running on this machine (the seventh, pdftoxml, successfully extracted text but failed to evaluate). It is harder to judge here which tool performed the best, as four tools had the best scores in two categories (though this is a bit misleading for LA-PdfText, as we will see).

After doing a rough score calculation, where the best result in a category was worth 6 points and the worst worth 1 (and ties getting the higher number), Icecite was the winner, with high marks across the board. It is especially impressive in its (relative) preservation of the correct number of new lines, and how very few misspelled words there are.

PdfBox was in second place, with the fewest missing new lines and paragraphs, and zero file errors. It was also the fastest of the six, with an average time of 8.8 seconds per file– if processing power is low, PdfBox is the best choice, as both Icecite and Grobid are resource-intensive.

Grobid was a close third, with on average no incorrectly ordered paragraphs and only 1 word missing per file. In fact, if time were no issue, Grobid would place second. (If time *is* an issue, however, Grobid is excruciatingly slow.)

By ranking of these criteria alone, pdfXtk seemingly did the worst, with pdf-extract not coming out much better. pdfXtk had a large amount of spurious paragraphs and words, as well as misspelled words– it ranked the worst in all three of these areas. Pdf-extract didn't score the worst on any criteria, but it had the second or third worst score on all measures except for extra words. However, I believe that neither of these were actually the worst tool, as both produced very readable output.

The same cannot be said for LA-PdfText. It had the fewest number of extra paragraphs, with on average only 5.1 per document, but a look at the score for missing paragraphs shows why this score is so high. On average, each file was missing 44 paragraphs: this was 65% of the average document. The criteria for missing and spurious words have a similar relationship; although there were only 0.2% of the words missing on average, most other tools had on average 0% missing. All of this becomes very obvious when inspecting the output, which is usually only a tiny fraction of what is in the actual document (and is often garbled at that). The output is worst in older documents; Appendix B shows the text output from LA-PdfText compared with the original PDF for a document from 2003 (Figure 3) and 2016 (Figure 4).

4.4 Comparison of results

It is difficult to assert very much about potential differences in the results from my Linux box environment and Bast and Korzen's results, as I was able to process only 197 files, 1.6% of the total 12,098 files available. Once evaluator.py is working on patas, I will be able to process all files using condor, which will give results that can be compared much more easily. However, I have made some preliminary observations based on the data processed.

Most measurements were comparable, percentage-wise. PdfBox had a huge amount more spurious newlines; 63% of newlines in the average document in my results versus merely 3.6% in theirs. LA-PdfText had very poor output in all results. pdfXtk gave me similar results to theirs, down to an error rate of about 6% files. For pdf-extract, my results were actually much better in all criteria except for error rate (3% of files in my results, versus 0.5%). I also had a much higher error rate for Grobid (3% vs 0.2%). The differences in error rate may be due to my setup or due to the fact that nearly half of the randomly selected folders I selected were from 2003 or earlier. More data will need to be processed to make any firm conclusions.

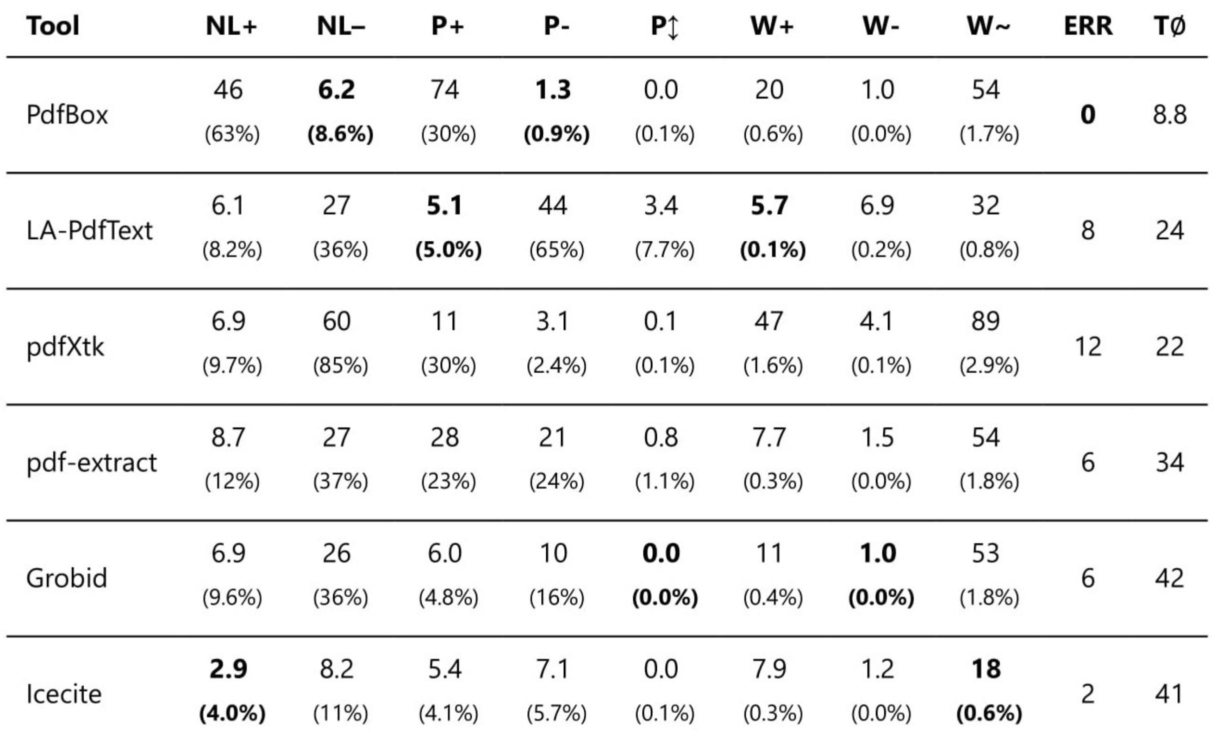
5 Further work needed

5.1 Further progress on tools

In time, I am certain that all 14 tools could get running on patas. The four that I could get running in a different Linux environment (LA-PdfText, pdf-extract, pdftoxml, and Grobid) obviously do work with the script, comparing what is different in the two environments should lead to success. The two that didn't work in any environment, pdf2xml and ParsCit, both are written in Perl and both have the same error message about a missing XML writer. Almost assuredly there is some sort of Perl package that could supply this.

5.2 Evaluation

The evaluation script works, but is incredibly picky and often crashes. (I have not gotten it to run successfully on patas.) The crash occurs when the script is unable to evaluate one of the tools. This causes a divide by zero error when it attempts to calculate an average. I have narrowed down the cause to a failure to find the output folder and/or to match



Tool	NL+	NL–	P+	P-	P↕	W+	W-	W~	ERR	TØ
PdfBox	46 (63%)	**6.2 (8.6%)**	74 (30%)	**1.3 (0.9%)**	0.0 (0.1%)	20 (0.6%)	1.0 (0.0%)	54 (1.7%)	**0**	8.8
LA-PdfText	6.1 (8.2%)	27 (36%)	**5.1 (5.0%)**	44 (65%)	3.4 (7.7%)	**5.7 (0.1%)**	6.9 (0.2%)	32 (0.8%)	8	24
pdfXtk	6.9 (9.7%)	60 (85%)	11 (30%)	3.1 (2.4%)	0.1 (0.1%)	47 (1.6%)	4.1 (0.1%)	89 (2.9%)	12	22
pdf-extract	8.7 (12%)	27 (37%)	28 (23%)	21 (24%)	0.8 (1.1%)	7.7 (0.3%)	1.5 (0.0%)	54 (1.8%)	6	34
Grobid	6.9 (9.6%)	26 (36%)	6.0 (4.8%)	10 (16%)	**0.0 (0.0%)**	11 (0.4%)	**1.0 (0.0%)**	53 (1.8%)	6	42
Icecite	**2.9 (4.0%)**	8.2 (11%)	5.4 (4.1%)	7.1 (5.7%)	0.0 (0.1%)	7.9 (0.3%)	1.2 (0.0%)	**18 (0.6%)**	2	41

Figure 1: Evaluation results run on fresh Linux install. NL^+: number of spurious newlines; NL^-: number of missing newlines; P^+: number of spurious paragraphs; P^-: number of missing paragraphs; $P^{\uparrow\downarrow}$: number of reordered paragraphs; W^+: number of spurious words; W^-: number of missing words; $W^\sim$: number of misspelled words. ERR shows the number of files that failed or timed out. TØ gives average time to process one file, in seconds. The best scores in each category are shown in bold.

the files to those in the groundtruth folder. This error can be triggered by renaming one of your folders. However, this still seems to occur on patas even when the extractor tool was run immediately before and the exact same output path was kept. This may in part be due to the folder structure; further work will need to be done to track down the source of the error.

5.3 New tools

In the future, new tools could potentially also be added to the list, for example pandoc, TextGrabber, or our department's own Freki. OCR-focused tools could also be added, such as Tesseract-OCR. Additionally, there are new tools being developed all the time, many utilizing deep learning techniques– both Tesseract and ParsCit have new versions that use neural nets.

6 Conclusion

In conclusion, text extraction from PDFs is far from a solved problem despite the format being around for over 25 years, with work on text extraction [going on] for nearly just as long. There are many factors that cause this, but most ultimately boil down to the way that information is encoded in a PDF file. While text extraction tools are always improving, even the best still are far from perfect and do not address all of the issues with the format.

References

[1] Adobe Systems Incorporated. *PDF Reference, version 1.7.* Author, San Jose, 6th edition, 2006.

[2] Apache, 2019. URL: `https://pdfbox.apache.org/`.

[3] Hannah Bast and Claudius Korzen. The Icecite research paper management system. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8181(2):396–409, 2013. `doi:10.1007/978-3-642-41154-0_30`.

[4] Hannah Bast and Claudius Korzen. A Benchmark and Evaluation for Text Extraction from PDF. In *Proceedings of Joint Conference On Digital Libraries*, Toronto, Ontario, Canada, 2017. `arXiv:arXiv:1709.01073v2`, `doi:10.1145/nnnnnnn.nnnnnnn`.

[5] Øyvind Raddum Berg. *High precision text extraction from PDF documents*. PhD thesis, University of Oslo, 2011.

[6] Øyvind Raddum Berg. Pdfextract, 2011. URL: `https://github.com/elacin/PDFExtract`.

[7] Steven Bird, Robert Dale, Bonnie J Dorr, Bryan Gibson, Mark T Joseph, Min-yen Kan, Dongwon Lee, Brett Powley, Dragomir R Radev, and Yee Fan Tan. The ACL Anthology Reference Corpus : A Reference Dataset for Bibliographic Research in Computational Linguistics. 2002.

[8] Gully Burns. La-pdftext, May 2013. URL: `https://github.com/GullyAPCBurns/lapdftext`.

[9] Ahmet Çakir. Usability and accessibility of portable document format. *Behaviour and Information Technology*, 35(4):324–334, 2016. `doi:10.1080/0144929X.2016.1159049`.

[10] Hervé Déjean and Emmanuel Giguet. pdf2xml, 2016. URL: `https://sourceforge.net/projects/pdf2xml`.

[11] T. Hassan and R. Baumgartner. Intelligent Text Extraction from PDF Documents. pages 2–6, 2006. `doi:10.1109/cimca.2005.1631436`.

[12] Tamir Hassan. Object-level document analysis of PDF files. page 47, 2009. `doi:10.1145/1600193.1600206`.

[13] Tamir Hassan. pdfxtk, Dec 2013. URL: `https://github.com/tamirhassan/pdfxtk`.

[14] Min-Yen Kan. Parscit, 2016. URL: `http://parscit.comp.nus.edu.sg/`.

[15] Claudius Korzen. Icecite, 2017. URL: `https://github.com/ckorzen/icecite`.

[16] M. Kruk. pdftohtml, 2013. URL: `https://sourceforge.net/projects/pdftohtml/`.

[17] Patrice Lopez. GROBID: Combining automatic bibliographic data recognition and term extraction for scholarship publications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5714 LNCS:473–474, 2009. `doi:10.1007/978-3-642-04346-8_62`.

[18] Patrice Lopez. Grobid, 2017. URL: `https://github.com/kermitt2/grobid`.

[19] W S Lovegrove and D F Brailsford. Document analysis of PDF files: methods, results and implications. *Electronic Publishing*, 8(3):207–220, 1995.

[20] Steven Moran and Michael Cysouw. *The Unicode Cookbook for Linguists orthography profiles*. Language Science Press, Berlin, 2018. `doi:10.5281/zenodo.1296780`.

[21] Derek Noonburg. Xpdf, 2019. URL: `https://www.xpdfreader.com/`.

[22] Yusuke Shinyama. Pdfminer, Sep 2016. URL: `https://github.com/euske/pdfminer`.

[23] Jörg Tiedemann. Improved Text Extraction from PDF Documents for Large-Scale Natural Language Processing. *Lecture Notes in Computer Science*, 8403(1):192–112, 2014.

[24] Jörg Tiedemann. pdf2xml, 2016. URL: `https://bitbucket.org/tiedemann/pdf2xml/`.

[25] Cornell University. arxiv.org e-print archive, 2017. URL: `https://arxiv.org/`.

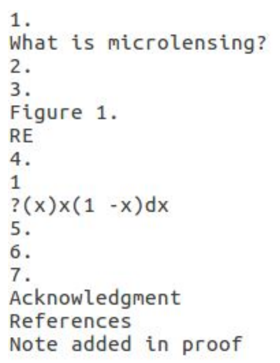
[26] Karl Jonathan Ward. pdf-extract, Sep 2015. URL: `https://github.com/CrossRef/pdfextract/`.

A Appendix A: Bast and Korzen's evaluation results

Tool	NL+	NL–	P+	P-	P↕	W+	W-	W~	ERR	TØ
pdftotext	14 (16%)	44 (53%)	60 (29%)	2.3 (0.6%)	1.4 (1.9%)	24 (0.7%)	2.4 (0.1%)	41 (1.2%)	2	**0.3**
pdftohtml	3.6 (4.3%)	70 (84%)	9.2 (31%)	4.2 (3.2%)	0.1 (0.1%)	16 (0.5%)	1.6 (0.0%)	95 (2.9%)	**0**	2.2
pdftoxml	33 (40%)	20 (25%)	80 (31%)	1.8 (0.5%)	0.1 (0.1%)	21 (0.6%)	1.5 (0.0%)	154 (4.7%)	1	0.7
PdfBox	**3.0 (3.6%)**	70 (85%)	7.6 (27%)	**0.9 (0.2%)**	0.0 (0.1%)	17 (0.5%)	**1.5 (0.0%)**	53 (1.6%)	2	8.8
pdf2xml	33 (40%)	39 (48%)	44 (21%)	40 (30%)	7.8 (9.5%)	8.6 (0.3%)	3.6 (0.1%)	34 (0.9%)	1444	37
ParsCit	15 (18%)	39 (47%)	10 (10%)	14 (6.4%)	1.3 (1.8%)	16 (0.5%)	2.3 (0.1%)	37 (1.1%)	1	6.8
LA-PdfText	5.5 (6.4%)	23 (28%)	**4.8 (3.1%)**	52 (73%)	2.9 (5.9%)	**5.7 (0.1%)**	6.1 (0.1%)	26 (0.6%)	324	24
PdfMiner	32 (38%)	18 (21%)	84 (30%)	3.6 (1.0%)	1.4 (2.1%)	34 (1.0%)	2.6 (0.1%)	110 (3.3%)	23	16
pdfXtk	7.9 (9.7%)	68 (84%)	12 (29%)	4.5 (3.5%)	0.1 (0.1%)	59 (1.8%)	6.1 (0.2%)	95 (3.0%)	739	22
pdf-extract	95 (114%)	53 (64%)	99 (32%)	8.4 (3.1%)	4.1 (7.7%)	74 (2.1%)	41 (1.2%)	149 (4.2%)	72	34
PDFExtract	9.5 (11%)	33 (40%)	28 (21%)	22 (25%)	0.8 (0.9%)	12 (0.4%)	2.8 (0.1%)	61 (1.8%)	176	46
Grobid	9.5 (11%)	30 (36%)	7.5 (6.7%)	11 (15%)	**0.0 (0.0%)**	14 (0.4%)	1.6 (0.0%)	63 (1.9%)	29	42
Icecite	3.4 (4.0%)	**10 (13%)**	6.2 (4.2%)	7.7 (5.5%)	0.1 (0.1%)	10 (0.3%)	1.7 (0.1%)	**21 (0.6%)**	34	41

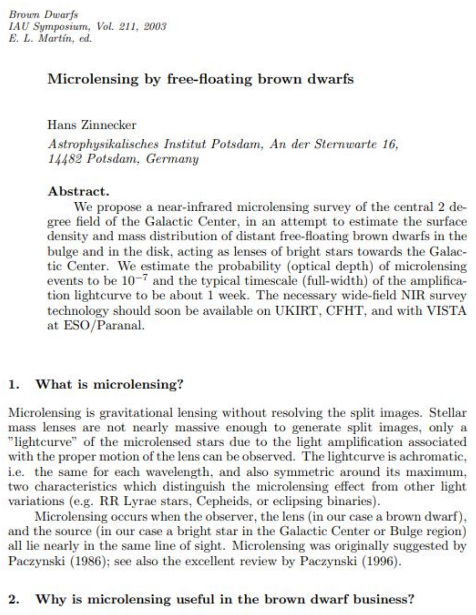
Figure 2: Summary of full evaluation results. NL^{+}: number of spurious newlines; NL^{-}: number of missing newlines; P^{+}: number of spurious paragraphs; P^{-}: number of missing paragraphs; $P^{\uparrow\downarrow}$: number of reordered paragraphs; W^{+}: number of spurious words; W^{-}: number of missing words; $W^{\sim}$: number of misspelled words. ERR shows the number of files that failed or timed out. TØ gives average time to process one file, in seconds. [4]

B Appendix B: Comparison of LA-PdfText output with PDF original



```
1.
What is microlensing?
2.
3.
Figure 1.
RE
4.
1
?(x)x(1 -x)dx
5.
6.
7.
Acknowledgment
References
Note added in proof
```

(a) Plain text output from LA-PdfText.



Brown Dwarfs
IAU Symposium, Vol. 211, 2003
E. L. Martín, ed.

Microlensing by free-floating brown dwarfs

Hans Zinnecker
Astrophysikalisches Institut Potsdam, An der Sternwarte 16, 14482 Potsdam, Germany

Abstract. We propose a near-infrared microlensing survey of the central 2 degree field of the Galactic Center, in an attempt to estimate the surface density and mass distribution of distant free-floating brown dwarfs in the bulge and in the disk, acting as lenses of bright stars towards the Galactic Center. We estimate the probability (optical depth) of microlensing events to be 10^{-7} and the typical timescale (full-width) of the amplification lightcurve to be about 1 week. The necessary wide-field NIR survey technology should soon be available on UKIRT, CFHT, and with VISTA at ESO/Paranal.

1. What is microlensing?

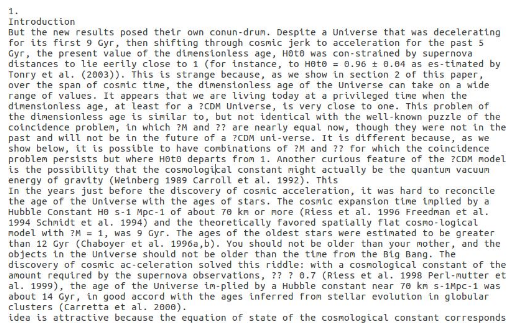
Microlensing is gravitational lensing without resolving the split images. Stellar mass lenses are not nearly massive enough to generate split images, only a "lightcurve" of the microlensed stars due to the light amplification associated with the proper motion of the lens can be observed. The lightcurve is achromatic, i.e. the same for each wavelength, and also symmetric around its maximum, two characteristics which distinguish the microlensing effect from other light variations (e.g. RR Lyrae stars, Cepheids, or eclipsing binaries).

Microlensing occurs when the observer, the lens (in our case a brown dwarf), and the source (in our case a bright star in the Galactic Center or Bulge region) all lie nearly in the same line of sight. Microlensing was originally suggested by Paczynski (1986); see also the excellent review by Paczynski (1996).

2. Why is microlensing useful in the brown dwarf business?

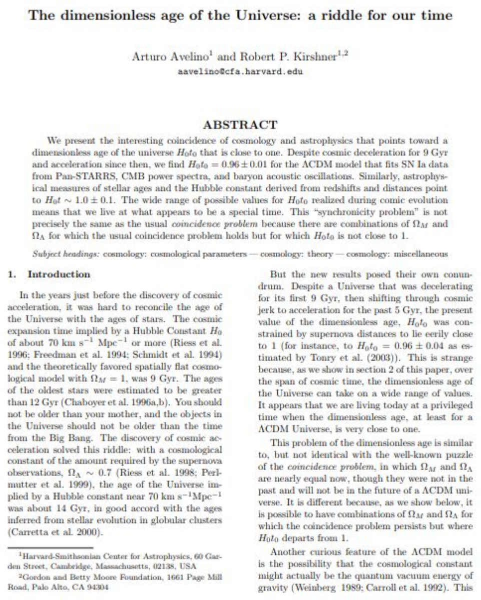
(b) First page of original PDF document.

Figure 3: Results from LA-PdfText on a document from 2003.



```
1.
Introduction
But the new results posed their own conun-drum. Despite a Universe that was decelerating
for its first 9 Gyr, then shifting through cosmic jerk to acceleration for the past 5
Gyr, the present value of the dimensionless age, H0t0 was con-strained by supernova
distances to lie eerily close to 1 (for instance, to H0t0 = 0.96 ± 0.04 as es-timated by
Tonry et al. (2003)). This is strange because, as we show in section 2 of this paper,
over the span of cosmic time, the dimensionless age of the Universe can take on a wide
range of values. It appears that we are living today at a privileged time when the
dimensionless age, at least for a ?CDM Universe, is very close to one. This problem of
the dimensionless age is similar to, but not identical with the well-known puzzle of the
coincidence problem, in which ?M and ?? are nearly equal now, though they were not in the
past and will not be in the future of a ?CDM uni-verse. It is different because, as we
show below, it is possible to have combinations of ?M and ?? for which the coincidence
problem persists but where H0t0 departs from 1. Another curious feature of the ?CDM model
is the possibility that the cosmological constant might actually be the quantum vacuum
energy of gravity (Weinberg 1989 Carroll et al. 1992). This
In the years just before the discovery of cosmic acceleration, it was hard to reconcile
the age of the Universe with the ages of stars. The cosmic expansion time implied by a
Hubble Constant H0 s-1 Mpc-1 of about 70 km or more (Riess et al. 1996 Freedman et al.
1994 Schmidt et al. 1994) and the theoretically favored spatially flat cosmo-logical
model with ?M = 1, was 9 Gyr. The ages of the oldest stars were estimated to be greater
than 12 Gyr (Chaboyer et al. 1996a,b). You should not be older than your mother, and the
objects in the Universe should not be older than the time from the Big Bang. The
discovery of cosmic ac-celeration solved this riddle: with a cosmological constant of the
amount required by the supernova observations, ?? ? 0.7 (Riess et al. 1998 Perl-mutter et
al. 1999), the age of the Universe im-plied by a Hubble constant near 70 km s-1Mpc-1 was
about 14 Gyr, in good accord with the ages inferred from stellar evolution in globular
clusters (Carretta et al. 2000).
idea is attractive because the equation of state of the cosmological constant corresponds
```

(a) Beginning of plain text output from LA-PdfText.



The dimensionless age of the Universe: a riddle for our time

Arturo Avelino[1] and Robert P. Kirshner[1,2]
aavelino@cfa.harvard.edu

ABSTRACT

We present the interesting coincidence of cosmology and astrophysics that points toward a dimensionless age of the universe H_0t_0 that is close to one. Despite cosmic deceleration for 9 Gyr and acceleration since then, we find $H_0t_0 = 0.96 \pm 0.01$ for the ΛCDM model that fits SN Ia data from Pan-STARRS, CMB power spectra, and baryon acoustic oscillations. Similarly, astrophysical measures of stellar ages and the Hubble constant derived from redshifts and distances point to $H_0t \sim 1.0 \pm 0.1$. The wide range of possible values for H_0t_0 realized during comic evolution means that we live at what appears to be a special time. This "synchronicity problem" is not precisely the same as the usual *coincidence problem* because there are combinations of Ω_M and Ω_Λ for which the usual coincidence problem holds but for which H_0t_0 is not close to 1.

Subject headings: cosmology: cosmological parameters — cosmology: theory — cosmology: miscellaneous

1. Introduction

In the years just before the discovery of cosmic acceleration, it was hard to reconcile the age of the Universe with the ages of stars. The cosmic expansion time implied by a Hubble Constant H_0 of about 70 km s^{-1} Mpc^{-1} or more (Riess et al. 1996; Freedman et al. 1994; Schmidt et al. 1994) and the theoretically favored spatially flat cosmological model with $\Omega_M = 1$, was 9 Gyr. The ages of the oldest stars were estimated to be greater than 12 Gyr (Chaboyer et al. 1996a,b). You should not be older than your mother, and the objects in the Universe should not be older than the time from the Big Bang. The discovery of cosmic acceleration solved this riddle: with a cosmological constant of the amount required by the supernova observations, $\Omega_\Lambda \sim 0.7$ (Riess et al. 1998; Perlmutter et al. 1999), the age of the Universe implied by a Hubble constant near 70 km $s^{-1}Mpc^{-1}$ was about 14 Gyr, in good accord with the ages inferred from stellar evolution in globular clusters (Carretta et al. 2000).

But the new results posed their own conundrum. Despite a Universe that was decelerating for its first 9 Gyr, then shifting through cosmic jerk to acceleration for the past 5 Gyr, the present value of the dimensionless age, H_0t_0 was constrained by supernova distances to lie eerily close to 1 (for instance, to $H_0t_0 = 0.96 \pm 0.04$ as estimated by Tonry et al. (2003)). This is strange because, as we show in section 2 of this paper, over the span of cosmic time, the dimensionless age of the Universe can take on a wide range of values. It appears that we are living today at a privileged time when the dimensionless age, at least for a ΛCDM Universe, is very close to one.

This problem of the dimensionless age is similar to, but not identical with the well-known puzzle of the *coincidence problem*, in which Ω_M and Ω_Λ are nearly equal now, though they were not in the past and will not be in the future of a ΛCDM universe. It is different because, as we show below, it is possible to have combinations of Ω_M and Ω_Λ for which the coincidence problem persists but where H_0t_0 departs from 1.

Another curious feature of the ΛCDM model is the possibility that the cosmological constant might actually be the quantum vacuum energy of gravity (Weinberg 1989; Carroll et al. 1992). This

[1]Harvard-Smithsonian Center for Astrophysics, 60 Garden Street, Cambridge, Massachusetts, 02138, USA

[2]Gordon and Betty Moore Foundation, 1661 Page Mill Road, Palo Alto, CA 94304

(b) First page of original PDF document.

Figure 4: Results from LA-PdfText on a document from 2016.