Quality comparison of python PDF text extraction libraries

Table of Contents

Abstract

This report presents a comprehensive analysis of Python libraries designed for PDF parsing, with the objective of identifying the optimal solution for our specific use case. The methodology involved rigorous testing of multiple libraries against a diverse collection of PDF documents, including both hand-made samples and publicly available files online. Each library was evaluated on its ability to accurately extract text content from the downloaded files, with performance measured against established ground truth data. Our assessment framework incorporated multiple quantitative and qualitative metrics to ensure a holistic evaluation of each library's capabilities, limitations, and overall suitability. The findings provide evidence-based recommendations for selecting the most appropriate PDF parsing solution that aligns with our technical requirements and operational constraints.

Motivation

As PDF text extraction constitutes a fundamental component of our project's user interface, selecting the most suitable Python library for this functionality is essential to our success. A methodical evaluation of available libraries will enable us to identify the solution that optimally addresses our specific requirements and performance needs.

This assessment is critical for several reasons. First, accurate text extraction directly impacts the user experience and the reliability of downstream data processing workflows. Second, different libraries offer varying levels of performance when handling complex PDF structures, embedded fonts, and multi-column layouts. Finally, factors such as processing speed, memory usage, and compatibility with our existing technology stack must be carefully weighed to ensure a sustainable implementation.

The findings from this evaluation will inform our technical decisions and provide a foundation for future maintenance and optimization efforts.

Methodology

To ensure the results of our experiments are accurate, we need to make sure that our methodology is accurate as well. Our methodology could include how we set up our testing, how we compare the Python libraries in question, and different metrics we use to represent the comparisons with scalable values.

Our methodology includes:

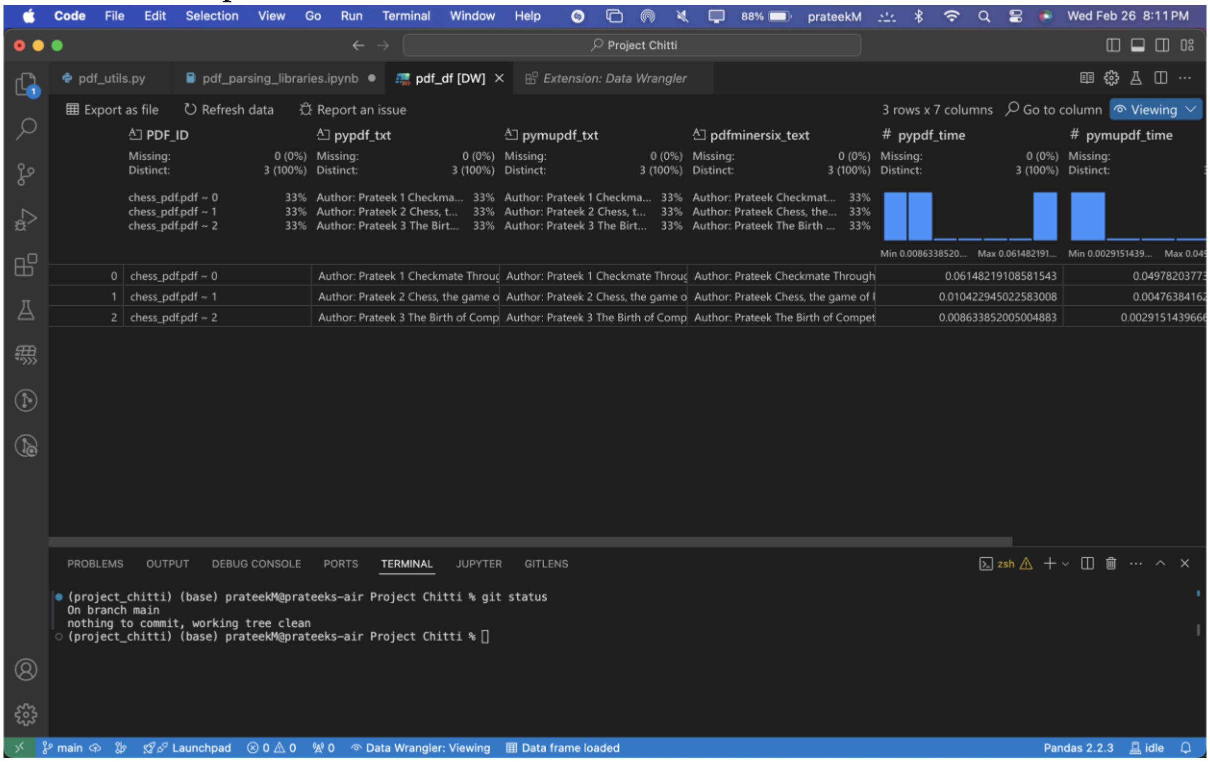
1) PDF Dataset
2) Ground Truth
3) Libraries under comparison
4) Metrics (both performance and accuracy metrics)

PDF Dataset

An important part of the PDF testing is being able to compare different elements of the extraction side-by-side, so that we have a visual representation of different elements of the different libraries. For example, some of the columns were “Performance”, “Time Taken”, or “Extracted Text”, so that we could compare how accurate/close they are when used by the different libraries tested. An image is attached at the end of this section of the report.

Another benefit of having a dataset of all the data collected is being able to create graphs so that we can visualize the data even better and compare the parameters in question.

This is an example of what the dataset would look like.



Code File Edit Selection View Go Run Terminal Window Help
88% prateekM Wed Feb 26 8:11 PM
Project Chitti
pdf_utils.py
pdf_parsing_libraries.ipynb
pdf_df [DW]
Extension: Data Wrangler
Export as file
Refresh data
Report an issue
3 rows x 7 columns
Go to column
Viewing
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL JUPYTER GITLENS
zsh
main Launchpad 0 0 0 Data Wrangler: Viewing Data frame loaded Pandas 2.2.3 idle

	PDF_ID	pypdf_txt	pymupdf_txt	pdfminersix_text	pypdf_time	pymupdf_time
	Missing: 0 (0%) Distinct: 3 (100%)	Missing: 0 (0%) Distinct: 3 (100%)	Missing: 0 (0%) Distinct: 3 (100%)	Missing: 0 (0%) Distinct: 3 (100%)	Missing: 0 (0%) Distinct: 3 (100%)	Missing: Distinct:
	chess_pdf.pdf ~ 0 33% chess_pdf.pdf ~ 1 33% chess_pdf.pdf ~ 2 33%	Author: Prateek 1 Checkma... 33% Author: Prateek 2 Chess, t... 33% Author: Prateek 3 The Birt... 33%	Author: Prateek 1 Checkma... 33% Author: Prateek 2 Chess, t... 33% Author: Prateek 3 The Birt... 33%	Author: Prateek Checkmat... 33% Author: Prateek Chess, the... 33% Author: Prateek The Birth ... 33%	Min 0.0086338520... Max 0.061482191...	Min 0.0029151439... Max 0.04
0	chess_pdf.pdf ~ 0	Author: Prateek 1 Checkmate Throug	Author: Prateek 1 Checkmate Throug	Author: Prateek Checkmate Through	0.06148219108581543	0.04978203773
1	chess_pdf.pdf ~ 1	Author: Prateek 2 Chess, the game o	Author: Prateek 2 Chess, the game o	Author: Prateek Chess, the game of	0.010422945022583008	0.00476384162
2	chess_pdf.pdf ~ 2	Author: Prateek 3 The Birth of Comp	Author: Prateek 3 The Birth of Comp	Author: Prateek The Birth of Compet	0.008633852005004883	0.0029151439666

```
(project_chitti) (base) prateekM@prateeks-air Project Chitti % git status
On branch main
nothing to commit, working tree clean
(project_chitti) (base) prateekM@prateeks-air Project Chitti % 
```

As you can see, this already represents the data more clearly, with columns such as time for different libraries mentioned, etc.

Ground Truth

The ground truth in this case is essentially everything the sample PDFs contain, all the text, put into a text file under the same name as the PDF.

The ground truth is important because this is one of the most important parts to how we can actually compare the accuracy of the actual text of the PDF and the extracted text from each of the libraries.

Libraries under comparison

Name	Lasy PyPi release	License	Version	Comments
Pypdf	February 9, 2025	BSD License	5.3.0	PyPdf2 is no longer maintained
Pymupdf	February 6, 2025	Dual Licensed - GNU AFFERO GPL 3.0 or Artifex Commercial License	1.25.3	
Pdfminer.six	July 6, 2024	MIT License (MIT)	20240706	

Metrics

Metrics are essentially the key method to finding out the accuracy of the extracted text using the libraries to the ground truth, which was discussed previously.

Our metric types include:

1) Semantic
2) Syntactic
3) Performance

Semantic Metrics

// ignore for now

Syntactic Metrics

This type of metric is what we essentially use to measure to accuracy – in this case, the accuracy between our "ground truth" and the extracted text from the libraries.

Our syntactic metrics are:

1) Levenshtein distance
2) BLEU Score
3) Jaccard similarity

a) Levenshtein distance:

Levenschtein distance is a way to measure how different two strings are. In simple terms. The way Levenschtein distance scores the difference is by measuring how many changes are needed to transform string 1 into string 2.

One of the pros of using Levenschtein distance is that it's relatively simple to use, as it just measures the number of changes needed to be made, so there's nothing extensively complicated involved with Levenschtein distance. Another pro of using this metric is that it's very flexible. This means that you can apply Levenschtein distance to different data types, like strings, or lists, etc.

One of the cons of using Levenschtein distance is for long strings, it can be very time-consuming (this is because the time complexity is O(m*n), which is a quadratic time complexity. Another con of using Levenschtein distance is sensitivity

to character order. This means that even though the overall meaning is the same, character order would skew the score because swapping order would count as change.
A working example of Levenschtein distance would he if **string 1** is "**kitten**", and **string 2** is "**sitting**". In this case, the Levenschtein distance would be 3, because there are 3 changes needed to be made for **string 1** to **string 2**. You need to substitute "k" with "s", substitute "e" with "i", and insert "g" at the end, which makes a total of 3 changes, which becomes our Levenschtein distance score.
A working example of Levenschtein distance would be if **string 1** is **"kitten"**, and **string 2** is **"sitting"**. This is how essentially how Levenschtein distance calculates the accuracy between the two:

1) Substitute "k" with "s" -- **string 1** is now "sitten"
2) Substitute "e" with "i" -- **string 1** is now "sittin"
3) Add "e" at the end – **string 1** is now "sitting".

<u>Therefore, the Levenschtein distance is **3.**</u>

b) BLEU Score:

The BLEU score is essentially a way to measure how good a machine translation is by comparing it to human translations. It looks at the words or groups of words (called n-grams) in the machine's translation and checks how many of them match the words in the reference translations. For example, if the machine translation has the same sequence of words as the reference translation, it gets a point for that. The more matches it has, the better the translation is scored.

A big advantage of BLEU is that it's quick and can be used to check many translations at once, making it useful for testing machine translations on a large scale. However, BLEU has some problems. It only counts exact matches of words, so if the machine uses different words with the same meaning, it might get a low score even if the translation is good. It also doesn't measure things like grammar or the overall meaning of the translation, which are also important for quality.

Here is a working example: Let's say that the original sentence we want to translate is: **"The cat sits on the mat."**. The reference translation is: **"The cat is sitting on the mat."**

1. The machine translation says: "The cat is sitting on the mat."
2. BLEU checks the words and phrases in the machine translation and compares them to the reference translation.
3. Since many of the words and/or word segments (like "The cat," "is sitting on," "on the mat") match exactly with the reference translation, BLEU gives a higher score.

c) Jaccard similarity

Jaccard similarity is basically a measure used to compare two sets and find out how similar they are. It's mostly used in areas like text analysis or recommendation systems, etc. The way it works is by comparing the number of items that both sets share (their intersection) to the total number of items in both sets combined (their union). To put this in a more mathematical format, the formula for Jaccard similarity is:

One major advantage of Jaccard similarity is its simplicity. It is easy to understand and calculate, making it great for quick comparisons between sets. Also, it's useful when you care only about the presence or absence of items, like comparing documents or lists of keywords. These advantages considered, there are some disadvantages: For example, this metric doesn't account for the frequency of elements in the sets. It also doesn't perform well when comparing very large sets, as it may overlook subtle (but important) differences that are important for more detailed analysis.

Here is a very simplified working example of Jaccard similarity: Let's say we have two sets, A and B:

- Set A is {apple, banana, cherry}
- Set B is {banana, cherry, grape}

Identify the intersection: The common items between A and B are {banana, cherry}.

Identify the union: All items from both sets are {apple, banana, cherry, grape}.

Calculate the size of the intersection: There are 2 items in the intersection.

Calculate the size of the union: There are 4 items in the union.

Performance Metrics

Performance metrics are important because it's lets us know how fast the PDF libraries actually extract the text and whatnot. Essentially, I just used the time module to measure the time between the start of the extraction process and the end of the process – this is for each page. This ensured that we got the time *only* for the extraction process and not anything else.

Results & Analysis

Semantic Metrics

// ignore for now

Syntactic Metrics

Levenshtein distance

// describe the result
// make sure we visualize the results with 1 or more graphs
A comparison of Levenshtein distance indicated the following:

a) pypdf outperformed both pymupdf &amp; pdfminer in terms of edit accuracy with a median of 109 edits per page to match the quality of Pipeline Builder text.

BLEU Score

Jaccard Similarity

Performance Metrics

Execution Time

Throughput

Conclusion for Project Chitti

Code

References (Citations)

1. https://pypi.org/project/pypdf/#description
2. https://pypi.org/project/PyMuPDF/#history
3. https://pypi.org/project/pdfminer.six/#history
4. https://drygast.net/blog/post/levenshtein_distance_in_db#:~:text=Levenshtein%20distance%20can%20be%20very,methods%20if%20performance%20is%20essential.

Future Work