DODFMiner

Release 1.3.7

KnEDLe Team

Nov 23, 2021

USER DOCUMENTATION

CHAPTER

ONE

INTRODUCTION

Official publications such as the Diário Oficial do Distrito Federal (DODF) are sources of information on all official government acts. Although these documents are rich in knowledge, analysing these texts manually by specialists is a complex and unfeasible task considering the growing volume of documents, the result of the frequent number of publications in the Distrito Federal Government's (GDF) communication vehicle.

This scenario is appropriate to employ computational techniques based on text mining and information visualization, in order to discover implicit and relevant knowledge in large textual data sets. It is known that these computational techniques receive data in a structured format. However, as DODF editions are originally published in unstructured format and in natural language, it is required to use techniques to prepare strategies in order to make the necessary adaptations to apply.

1.1 DODFMiner

With all that in mind, the DODFMiner is the software that is being developed for the extraction of data from documents in PDF format referring to the publications of the Official Gazette of the Federal District, Brazil.

CHAPTER

TWO

INSTALLATION

Table of Contents

DODFMiner is currently only supported on Linux and OSX. It may be possible to install on Windows, though this hasn't been extensively tested.

2.1 Requirements

- Python3
- MuPDF

2.2 Installing MuPDF

MuPDF is the main engine used to parse pdf files on DODFMiner. Its installation is essencial for proper work.

2.2.1 macOS

In macOS use brew to install the library:

```
$ brew install mupdf
```

2.2.2 Debian Linux (Ubuntu)

On Ubuntu, or other Debian Linux distro, use the following commands:

```
$ add-apt-repository ppa:ubuntuhandbook1/apps
$ apt-get update
$ apt-get install mupdf mupdf-tools
```

2.3 DODFMiner Installation Methods

We support two method of installation. The Library method (recommended), and a Docker Install.

2.3.1 Library Install

From The Python Package Index (PyPI):

```
pip install dodfminer
```

From Github:

```
git clone https://github.com/UnB-KnEDLe/DODFMiner.git
cd dodfminer
pip install -e .
```

2.3.2 Docker Install

Since this project have several dependencies outside Python libraries, there is a DockerFile and a Compose file provided to facilitate the correct execution. The DockerFile contains instructions on how the image is build, while the Compose file contains instruction on how to run the image.

The container created by the DockerFile image use a DATA_PATH environment variable as the location to save the downloaded DODF PDFs and the extracted JSONs. This variable needs to be set before the execution.

To build and execute the image the docker and docker-compose need to be correct installed:

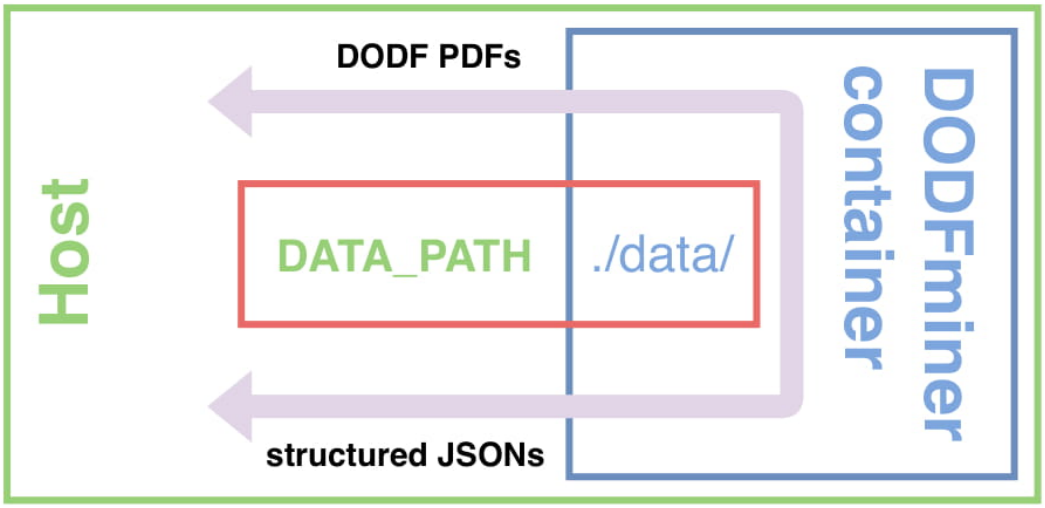
1. Install Docker
2. Install Docker Compose

After the installation, the first thing that docker needs is an image. To create the image run the following command in the root of the project:

```
$ docker-compose build
```

This can took a while to finish.

Now, with the image created, the docker-compose can generate instances (containers) of this image to run specifics tasks.




```
$ export DATA_PATH=/path/to/save/files/ \
$ sudo -E docker-compose run dodfminer -sd 01/19 -ed 01/19
```

This command executes the download task, where -st is the start date and -ed is the end date, representing the interval that the DODFs will be downloaded.

Other arguments can be found excuting the command:

```
$ export DATA_PATH=/path/to/save/files/ \
$ sudo -E docker-compose run dodfminer --help
```

Note: 1. If your docker is already in the _sudo_ group you can execute without _sudo_, otherwise the -E argument is needed for _sudo_ use the environment variables declared in login _bash_.

2. The container will not work if the DATA_PATH is not defined in the environment.

CHAPTER

THREE

USING DODFMINER

Table of Contents

3.1 Command-Line Usage

Considering the module has been installed using pip, you should be able to use DODFMiner as a command line program. To check if installation has been done successfully run:

```
$ dodfminer --help
```

A help screen of the program should appear. The helper sould show two positional arguments: *downloader* and *extract*. Each of those arguments can be considered as a subprogram and work independently, you can choose the one you desire using:

```
$ dodfminer downloader --help
$ dodfminer extract --help
```

Depending which module you choose the execution parameters will change.

3.1.1 Downloader Module

The downloader module is responsible for downloading DODF PDFs from the website. It allows you to choose the start and end date of the files you want to download. Also, you can choose where to save them. Following are the list of avaiable parameters, their description and the default value.

Note: This module relies on internet connection and can fail if internet is not working properly. Also, the execution might take a while if there are a huge ammount of pdfs to download.

Parameters Table

Argument	Description	Default
-sp –save_path	Folder to output the download DODFs	./
-sd –start_date	Input the date in either mm/yyyy or mm-yyyy	01/2019
-ed –end_date	Input the date in either mm/yyyy or mm-yyyy	01/2019

Usage Example:

```
$ dodfminer downloader -sd 01/2003 -ed 05/2004
```

3.1.2 Extractor Module

The extractor module is responsible for extracting information from DODF PDFs and save it in a desirable format.

The extraction can be made, to a pure text content, where a DODF will be converted to TXT or JSON. Or, additionaly, the extraction can be done in a polished way, where from the DODF will be extracted to acts and its given proprieties in a CSV format.

Pure extraction

Given a -t flag, it allows you to choose the output format between three options: blocks of text with tiles, pure text in .txt format and text separated by titles:

- **Blocks of Text**: Outputs a JSON file that extract text blocks.
- **Pure Text**: Output a .txt file, with raw text from the pdf.
- **Blocks of Text with Titles**: Outputs a JSON file that extract text blocks indexed by titles.

Polished Extraction

Using the -a or –act flag, you can extract the dodf in a polished way. The usage of the -a will extract all types of act in the DODF. Additionaly, if desired, the flag can followed by a list of specific acts types which you want to extract. The extraction is done using the backend specified in the -b flag, which can be either regex or ner.

Available Act Types:

- aposentadoria
- reversoes
- nomeacao

- exoneracao
- abono
- retificacoes
- substituicao
- cessoes
- sem_efeito_aposentadoria
- efetivos_nome
- efetivos_exo

Parameters Table

Following are the list of avaiable parameters, their description and the default value.

Argument	Description	Default
-i –input_folder	Path to the PDFs folder	./
-s –single-file	Path to a single PDF	None
-t –type-of-extraction	Type of text extraction	None
-a –act	List of acts that will be extract to CSV	all
-b –backend	Which backend will extract the acts	regex

Usage Example:

```
$ dodfminer extract -i path/to/pdf/folder -t with-titles
$ dodfminer extract -s path/to/dodf.pdf -t pure-text
$ dodfminer extract -s path/to/dodf.pdf -a nomeacao
$ dodfminer extract -s path/to/dodf.pdf -a nomeacao cessoes -b ner
```

Note: It's important to notice that if -t and -a options are used together the -t option will have the priority and the -a will not execute.

Note: The DODFMiner act extraction needs the text data from DODFs to correct extract the acts from DODF, therefore the -a option generates first txt files before the act extraction.

3.2 Library Usage

The DODFMiner was created also thinking the user might want to use it as a library in their own projects. Users can use install the DODFMiner and call its modules and functions in their python scripts. Following are some of the imports you might want to do, while using as a library:

```
from dodfminer import acts
from dodfminer import Downloader
from dodfminer import ActsExtractor
from dodfminer import ContentExtractor
```

The details of using the DODFMiner modules and functions are described in this documentation, in the following sections.

CHAPTER

FOUR

ARCHITECTURE'S DOCUMENT

Python is surprisingly flexible when it comes to structuring your applications. On the one hand, this flexibility is great: it allows different use cases to use structures that are necessary for those use cases. On the other hand, though, it can be very confusing to the new developer.

4.1 Document Overview

4.2 Introduction

4.2.1 Objetive

This document aims to provide an overview of the architecture of the DODFMiner Library: it contains pertinent information about the architecture model adopted, such as diagrams that illustrate use cases, package diagram, among other resources.

4.2.2 Escope

Through this document, the reader will be able to understand the functioning of the DODFMiner Library, as well as the approach used in its development. In this way, it will be possible to have a broad understanding of its architecture.

4.2.3 Definitions, Acronyms and Abreviations

4.2.4 Revision History

4.3 Architectural Representation

The main point to understand in this architecture is that the DODFMiner is a library and a CLI application simultaniously. DODFMiner can be integrated to another project or used standalone in a shell environment.

Being a library requires a given ammount of complexity. In larger applications, you may have one or more internal packages that provide specific functionality to a larger library you are packaging. This application follows this aspect, mining pdf documents, imply in many subpackages with specific functionality, that when working together, fulfill a greater aspect.

4.3.1 Relationship Diagram

4.3.2 Subpackages Structure

This applications follow the basic structure for a python library with multiple subpackages. It uses a common concept of *core* and *helper* files.

The *core* file is the main file in a package or subpackage, it contains the class with the main package execution. The *helper* file contains suporting functions to the package.

In summary, the project structure look as follows:

```
dodfminer
├── __version__.py
├── cli.py
├── downloader
│   └── core.py
├── extract
│   ├── polished
│   │   ├── acts
│   │   │   ├── abono.py
│   │   │   ├── aposentadoria.py
│   │   │   ├── base.py
│   │   │   ├── cessoes.py
│   │   │   ├── exoneracao.py
│   │   │   ├── models/
│   │   │   ├── nomeacao.py
│   │   │   ├── reversoes.py
│   │   │   ├── sem_efeito_aposentadoria.py
│   │   │   └── substituicao.py
│   │   ├── backend
│   │   │   ├── ner.py
│   │   │   └── regex.py
│   │   ├── core.py
│   │   └── helper.py
│   └── pure
│       ├── core.py
│       └── utils
│           ├── box_extractor.py
│           ├── title_extractor.py
│           └── title_filter.py
└── run.py
```

4.3.3 Technologies

Following are some of the most essencial tecnologies used with the DODFMiner application

1. **MuPDF**

   MuPDF is a free and open-source software framework written in C that implements a PDF, XPS, and EPUB parsing and rendering engine. It is used primarily to render pages into bitmaps, but also provides support for other operations such as searching and listing the table of contents and hyperlinks

2. **BeautifulSoup**

   Beautiful Soup is a Python package for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML, which is useful for web scraping

3. **Pandas**

   Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license

4. Site do DODF

   Website where all of the DODFs are downloaded from.

4.4 Goals and Constraints

4.4.1 Non-functional Requirements

- Be a library avaiable by pip on The Python Package Index (PyPI)
- Work as a standalone command line application, installed globally without needing file execution
- Support continuous deployment and continuous integration
- The DODFMiner should be able to:
  - Download DODFs from the website
  - Extract pdf files to .txt and .json formats
  - Extract images and tables from the DODF
  - Extract DODF's Acts and its proprieties to a dataframe or other desirable format

4.4.2 General Constraints

- Have tested support for Mac and Linux users.
- Have a docker installation method
- Be open-source
- Don't use a database library

4.4.3 Tecnological Constraints

- Python: Language used for development
- MuPDF: Tool used for PDF extraction
- BeautifulSoup: Library used for webscraping
- Pandas: Library used for data handling and cration of dataframes
- DODF Website: Website in which the DODFs are downloaded from

4.5 Logical View

4.5.1 Overview

DODFMiner is a library and CLI application made with the Python language, using MuPDF, BeautifulSoup, Pandas, and many others python libraries. The purpose of DODFMiner is to be an library and tool to fullfil the hole process of extraction of a official diary from federal district in Brazil.

4.5.2 Package Diagram

4.5.3 Class Diagram

4.6 References

Amika Architecture

Python Layouts

CHAPTER

FIVE

CODE OF CONDUCT

5.1 Purpose

5.2 Our standards

5.2.1 Expected behavior

- Participate in an authentic and active way. In doing so, you contribute to the health and longevity of this community.
- Consider respect in your speech and actions.
- Try to collaborate before the conflict.
- Refrain from degrading, discriminatory or harassing behavior.
- Be aware of your surroundings and fellow participants. Alert community leaders if you notice a dangerous situation, someone in danger or violations of this Code of Conduct.
- Remember that community event venues can be shared with others. Be respectful to all regulars or customers.

5.2.2 Unacceptable behavior

- Violence, threats of violence or aggressive language directed against another person.
- Sexist, racist, LGBTphobic, or otherwise discriminatory speeches.
- Publication or exhibition of sexually explicit or violent material.
- Publication or threat of publication of personally identifiable information.
- Personal insults, particularly those related to gender, sexual orientation, ethnicity, religion or disability.
- Inadequate photography or recording.
- Inappropriate physical contact. You must have someone's consent before touching it.
- Bullying. Exposing someone to humiliating and embarrassing, repetitive situations. Psychological violence, intimidation, harassment (online or in person).
- Sexual harassment. This includes sexual comments or jokes; inappropriate movements, unsuccessful attempts and sexual advances.
- Advocate or encourage any of the above behaviors.
- Continuous interruption of community events, including lectures and presentations.

5.3 Our Responsibilities

5.4 Enforcement

CHAPTER

SIX

CONTRIBUTING GUIDE

6.1 How to contribute?

To contribute with this project, you just need to follow the steps up next

- *Fork* of the repository (for external users only)
- Create branchs
- Follow the commits policy
- Submit Pull Request

6.2 Branch Policy

6.2.1 main

The **main** branch is the production branch, where the stable version of the project will be. It will be blocked for commits and pushs. See the merges policy in the topic Merges to **main**.

6.2.2 development

The **dev** branch is where the work of the other branches will be unified and where a stable version will be created to merge with **main**. Like **main** it is blocked for commits and pushs. See the merges policy in the topic [Merges for dev] (CONTRIBUTING.md#merges-for-development) merges to **dev** .

6.2.3 Branch name

The feature development branches will be created from the **dev** branch with the default nomenclature `change_name`.

6.3 Commits Policy

Commits must be made using the `-s` parameter to indicate your signature on the commit.

```
git commit -s
```

Also, for double commits the `-s` command must be used, and the signature of your pair must be added.

The commit comment must be in english and show the action taken, or the change made.

Comment of commit:

```
Making contribution guide

Change detail

Signed-off-by: Isaque Alves <isaquealvesdl@gmail.com>
Signed-off-by: Felipe Campos <fepas.unb@gmail.com>
```

In order for both involved in the commit to be included as contributors in the GitHub commits graph, just include the statement `Co-authored-by:` in the message:

```
Making contribution guide

Signed-off-by: Isaque Alves <isaquealvesdl@gmail.com>
Signed-off-by: Felipe Campos <fepas.unb@gmail.com>

Co-authored-by: Isaque Alves <isaquealvesdl@gmail.com>
Co-authored-by: Felipe Campos <fepas.unb@gmail.com>
```

For commits that include a small change that has already been resolved, start the commit message with `HOTFIX <message>`

Example of a commit comment:

```
HOTFIX Updating project contribution guide
```

6.4 Merges and Pull Requests Policy

6.4.1 Pull Requests

Pull requests must be made to the **dev** branch following the rules and steps in the topic **Merges**. In the pull request content there should be a clear description of what was done.

Work in Progress

If there is a need to update the **main** branch before completing the issue, the name of the pull request must contain WIP: <ran_name> so that the branch is not deleted.

6.5 Merges

Merges to **main** should be made when the functionality or refactoring is in accordance with the following aspects:

- Functionality or refactoring completed;
- **Build** of Travis passing;
- Progress or maintain the percentage of test coverage;
- Functionality reviewed by some other member.

To merge into **main** the steps to be followed are:

- `git checkout branch_of_work`;
- `git pull --rebase origin main`;
- `git push origin branch_of_work`;
- Open pull request via GitHub interface;
- Wait for Code Review

6.5.1 Code Review

The code review must be done by one or more team members who did not participate in the changes. After at least a Code Review, Status Check (Travis, CodeClimate) approval, PullRequest can be accepted;

To accept PullRequest, you must use the **merge** option on Github.

6.6 Test Coverage

Code coverage is constantly evaluated and the goal is that it never decreases. "Tested code generates less rework and more quality of life".

CHAPTER

SEVEN

DOWNLOADER CORE

Table of Contents

Download DODFs from the Buriti Website and save on proper directory.

Download monthly pdfs of DODFs.

Usage example:

```
downloader = Downloader()
downloader.pull(start_date, end_date)
```

7.1 Downloader Class

class dodfminer.downloader.core.**Downloader**(*save_path='./'*)

Responsible for the download of the DODFs Pdfs.

Parameters **save_path** (*str*) – Path to save the downloads.

_download_path

Folder in which the downloads will be stored.

_prog_bar

Indicate if download should contain a progress bar.

pull(*start_date*, *end_date*)

Make the download of the DODFs pdfs.

All dodfs are downloaded from start_date to end_date inclusively. The Pdfs are saved in a folder called "data" inside the project folder.

Parameters

- **start_date** (*str*) – The start date in format mm/yyyy.
- **end_date** (*str*) – The start date in format mm/yyyy.

Note: The name or the path of the save folder are hard coded and can't be changed due to some nonsense software engineer decision.

7.2 Downloader Private Methods

One does not access directly none of those methods, but they are listed here in case the programmer using the downloader library needs more informations.

7.2.1 Path Handling

Methods that handle the creation of the paths to the dowloaded DODFS.

Downloader.**_create_single_folder**(*path*)
Create a single folder given the directory path.

This function might create a folder, observe that the folder already exists, or raise an error if the folder cannot be created.

Parameters **path** (*str*) – The path to be created

Raises **OSError** – Error creating the directory.

Downloader.**_create_download_folder**()
Create Downloaded DODFs Structures.

Downloader.**_make_month_path**(*year*, *actual_date*)
Create and return the folder for the year and month being download.

Parameters

- **year** (*int*) – The year respective to the folder.
- **actual_date** (datetime) – The date in which the downloaded
- **corresponds.** (*DODF*) –

Returns The path to the actual month in which the download is being made.

7.2.2 URL Making

Methods that construct an URL to further make the download request. .. .. automethod:: dodfminer.downloader.core.Downloader._make_url .. .. automethod:: dodfminer.downloader.core.Downloader._make_href_url .. .. automethod:: dodfminer.downloader.core.Downloader._make_download_url

7.2.3 Web Requests

Methods that handle the download request and its execution.

Downloader._fail_request_message(*url*, *error*)
Log error messages in download.

Parameters

- **url** (*str*) – The failing url to the website.
- **error** (*str*) – The kind of error happening.

Downloader._download_pdf(*url*, *path*)
Download the DODF PDF.

Note: Might be time consuming depending on bandwidth.

Parameters

- **url** (*str*) – The pdf url.
- **path** (*str*) – The path to save the pdf.

Raises **RequestException** – Error in case the request to download fails.

7.2.4 Others

Other methods for the downloader library.

classmethod Downloader._string_to_date(*date*)
Convert the date to datetime.

Parameters **date** (datetime) – The date to be converted in string format.

Returns Return the date formated in string now as datetime datatype.

Raises **Exception** – date passed through cli is in wrong format.

Downloader._file_exist(*path*)
Check if a file exists.

Prevents redownloads.

Parameters **path** (*str*) – The path where the file might be

Returns Boolean indicating if file does really exists.

Downloader._log(*message*)
Logs a message following the downloader pattern.

Parameters **message** (*str*) – The message to be logged.

CHAPTER

EIGHT

PURE CORE

Table of Contents

Extract content from DODFS and export to JSON.

Contains class ContentExtractor which have to public functions avaiable to extract the DODF to JSON

Usage example:

```
from dodfminer.extract.pure.core import ContentExtractor

pdf_text = ContentExtractor.extract_text(file)
ContentExtractor.extract_to_txt(folder)
```

8.1 Extract Class

class dodfminer.extract.pure.core.**ContentExtractor**

Extract content from DODFs and export to JSON.

Extracts content from DODF files using as suport the title and subtitle databases—which runs using MuPDF—, and the Tesseract OCR library. All the content is exported to a JSON file, in which its keys are DODF titles or subtitles, and its values are the correspondent content.

Note: This class is not constructable, it cannot generate objects.

---

classmethod extract_structure(*file*, *single=False*, *norm='NFKD'*)

Extract boxes of text with their respective titles.

Parameters

- **file** – The DODF file to extract titles from.
- **single** – Output content in a single file in the file directory.
- **norm** – Type of normalization applied to the text.

Returns

A dictionaty with the blocks organized by title.

Example:

```
{
    "Title": [
        [
            x0,
            y0,
            x1,
            y1,
            "Text"
        ]
    ],
    ...
}
```

classmethod extract_text(*file*, *single=False*, *block=False*, *is_json=True*, *sep=' '*, *norm='NFKD'*)

Extract block of text from file

Parameters

- **file** – The DODF to extract titles from.
- **single** – output content in a single file in the file directory.
- **block** – Extract the text as a list of text blocks.
- **json** – The list of text blocks are written as a json file.
- **sep** – The separator character between each block of text.
- **norm** – Type of normalization applied to the text.

Note: To learn more about the each type of normalization used in the *unicode.normalization* method, click here.

Returns

These are the outcomes for each parameter combination.

When *block=True* and *single=True*: In case *json=True*, The method saves a JSON file containing the text blocks in the DODF file. However, is case *json=False*, the text from the whole PDF is saved as a string in a .txt file.

When *block=True* and *single=False*: The method returns an array containing text blocks.

Each array in the list have 5 values: the first four are the coordinates of the box from where the text was extracted (x0, y0, x1, y1), while the last is the text from the box.

Example:

```
(127.77680206298828,
194.2507781982422,
684.0039672851562,
211.97523498535156,
"ANO XLVI EDICAO EXTRA No- 4 BRASILIA - DF")
```

When *block=False* and *single=True*: The text from the whole PDF is saved in a .txt file as a normalized string.

When *block=False* and *single=False*: The method returns a normalized string containing the text from the whole PDF.

classmethod extract_to_json(*folder='./'*, *titles_with_boxes=False*, *norm='NFKD'*)

Extract information from DODF to JSON.

Parameters

- **folder** – The folder containing the PDFs to be extracted.
- **titles_with_boxes** – If True, the method builds a dict containing a list of tuples (similar to *extract_structure*).
- **Otherwise** (similar to *extract_text*) –
- **tuples** (*the method structures a list of*) –
- **norm** – Type of normalization applied to the text.

Returns For each PDF file in data/DODFs, extract information from the PDF and output it to a JSON file.

classmethod extract_to_txt(*folder='./'*, *norm='NFKD'*)

Extract information from DODF to a .txt file.

For each PDF file in data/DODFs, the method extracts information from the PDF and writes it to the .txt file.

Parameters

- **folder** – The folder containing the PDFs to be extracted.
- **norm** – Type of normalization applied to the text.

8.2 Extractor Private Members

One does not access directly none of those methods, but they are listed here in case the programmer using the extract library needs more informations.

8.2.1 Text Preprocessing

classmethod ContentExtractor.**_normalize_text**(*text*, *form='NFKD'*)
This method is used for text nomalization.

Parameters

- **text** – The text to be normalized.
- **form** – Type of normalization applied to the text.

Returns A string with the normalized text.

classmethod ContentExtractor.**_extract_titles**(*file*)
Extract titles and subtitles from the DODF.

Parameters **file** – The DODF to extract the titles.

Returns

An object of type ExtractorTitleSubtitle, in which have the attributes:

titles: get all titles from PDF. subtitle: get all subtitles from PDF.

Raises **Exception** – error in extracting titles from PDF.

8.2.2 Check Existence

classmethod ContentExtractor.**_get_pdfs_list**(*folder*)
Get DODFs list from the path.

Parameters **folder** – The folder containing the PDFs to be extracted.

Returns A list of DODFS' PDFs paths.

classmethod ContentExtractor.**_get_json_list**(*folder*)
Get list of exisiting JSONs from the path.

Parameters **folder** – The folder containing the PDFs to be extracted.

Returns A list of all exisiting JSONs.

classmethod ContentExtractor.**_get_txt_list**(*folder*)
Get list of exisiting .txt files from the path.

Parameters **folder** – The folder containing the PDFs to be extracted.

Returns A list of all exisiting .txt files.

8.2.3 Directory Handling

classmethod ContentExtractor.**_struct_subfolders**(*path*, *json_f*, *folder*)
Creates a directory for the JSON files.

This method structures the folder tree for the allocation of files the code is curretly dealing with.

Parameters

- **path** – The path to the extracted file.
- **json_f** (*boolean*) – If True, the file will extracted to a JSON. Otherwise, it will be extrated to a .txt.

- **folder** – The folder containing the PDFs to be extracted.

Raises **FileExistsError** – The folder being created is already there.

Returns The path created for the JSON to be saved.

classmethod ContentExtractor.**_create_single_folder**(*path*)

Create a single folder given the directory path.

This function might create a folder, observe if the folder already exists, or raise an error if the folder cannot be created.

Parameters **path** – The path to be created.

Raises **OSError** – Error creating the directory.

8.2.4 Others

classmethod ContentExtractor.**_log**(*msg*)

Print message from within the ContentExtractor class.

Parameters **msg** – String with message that should be printed out.

CHAPTER

NINE

PURE UTILS

Warning: This documentation needs improvments by the code's author.

Table of Contents

9.1 Box Extactor

Functions to extract boxes from text.

dodfminer.extract.pure.utils.box_extractor.**compare_blocks**(*block1*, *block2*)

Implements a comparison heuristic between blocks. Blocks that are in the uppermost and leftmost positions should be inserted before the other block in comparison.

Parameters

- **block1** – a block tuple to be compared.
- **block2** – a block tuple to be compared to.

Returns Int

dodfminer.extract.pure.utils.box_extractor.**draw_doc_text_boxes**(*doc: fitz.Document*, *doc_boxes*, *save_path=None*)

Draw extracted text blocks rectangles. In result, a pdf file with rectangles shapes added, representing the extracted blocks, is saved.

Parameters

- **doc** – an opened fitz document

- **doc_boxes** – the list of blocks on a document, separated by pages
- **save_path** – a custom path for saving the result pdf

Returns None

dodfminer.extract.pure.utils.box_extractor.**get_doc_img_boxes**(*doc: fitz.Document*)
Returns list of list of bouding boxes of extracted images.

Parameters **doc** – an opened fitz document

Returns

List[List[Rect(float, float, float, float)]]. Each Rect represents an image bounding box.

dodfminer.extract.pure.utils.box_extractor.**get_doc_text_boxes**(*doc: fitz.Document*)
Returns list of list of extracted text blocks.

Parameters **doc** – an opened fitz document.

Returns List[List[tuple(float, float, float, float, str, int, int)]]

dodfminer.extract.pure.utils.box_extractor.**get_doc_text_lines**(*doc: fitz.Document*)
Returns list of list of extracted text lines.

Parameters **doc** – an opened fitz document.

Returns List[List[tuple(float, float, float, str)]]

dodfminer.extract.pure.utils.box_extractor.**sort_blocks**(*page_blocks*)
Sort blocks by their vertical and horizontal position.

Parameters **page_blocks** – a list of blocks within a page.

Returns List[tuple(float, float, float, float, str, int, int)]

dodfminer.extract.pure.utils.box_extractor.**_extract_page_lines_content**(*page*)
Extracts page lines.

Parameters **page** – fitz.fitz.Page object to have its bold content extracted.

Returns List[tuple(float, float, float, float, str)] A list containing lines content at the page, along with its bounding boxes.

dodfminer.extract.pure.utils.box_extractor.**get_doc_text_boxes**(*doc: fitz.Document*)
Returns list of list of extracted text blocks.

Parameters **doc** – an opened fitz document.

Returns List[List[tuple(float, float, float, float, str, int, int)]]

dodfminer.extract.pure.utils.box_extractor.**get_doc_text_lines**(*doc: fitz.Document*)
Returns list of list of extracted text lines.

Parameters **doc** – an opened fitz document.

Returns List[List[tuple(float, float, float, str)]]

dodfminer.extract.pure.utils.box_extractor.**_get_doc_img**(*doc: fitz.Document*)
Returns list of list of image items.

Note: This function is not intented to be used by final users, but internally. Image *items* are described at:

https://pymupdf.readthedocs.io/en/latest/page/#Page.getImageBbox

---

Parameters doc – an opened fitz document

Returns List[List[tuple(int, int, int, int, str, str, str, str, int)]] (xref, smask, width, height, bpc, colorspace, alt. colorspace, filter, invoker)

dodfminer.extract.pure.utils.box_extractor.**get_doc_img_boxes**(*doc: fitz.Document*)
Returns list of list of bouding boxes of extracted images.

Parameters doc – an opened fitz document

Returns

List[List[Rect(float, float, float, float)]]. Each Rect represents an image bounding box.

9.2 Title Filter

Find titles using a Filter.

class dodfminer.extract.pure.utils.title_filter.**BoldUpperCase**
Filter functions useful for bold and upper case text.

Note: This class is static and should not be instanciated.

classmethod dict_bold(*data*)
Hmm.

Evaluates do True if d['flags'] matches the following conditions:

- is one of the values in BoldUpperCase.BOLD_FLAGS

classmethod dict_text(*data*)
Check if text is title.

Evaluates to true if d['text'] matches the following conditions:

- all letters are uppercase;
- does not contain 4 or more consecutive spaces;
- has a len greater than BoldUpperCase.TEXT_MIN/

Returns Boolean indicating if text is title.

9.3 Title Extactor

Extract Title and Subtitles.

class dodfminer.extract.pure.utils.title_extractor.**BBox**(*bbox*)

property bbox
Alias for field number 0

class dodfminer.extract.pure.utils.title_extractor.**Box**(*x0*, *y0*, *x1*, *y1*)

property x0
Alias for field number 0

property x1
Alias for field number 2

property y0
Alias for field number 1

property y1
Alias for field number 3

class dodfminer.extract.pure.utils.title_extractor.ExtractorTitleSubtitle(*path*)
Use this class like that: >> path = "path_to_pdf" >> extractor = ExtractorTitleSubtitle(path) >> # To extract titles >> titles = extractor.titles >> # To extract subtitles >> titles = extractor.subtitles >> # To dump titles and subtitles on a json file >> json_path = "valid_file_name" >> extractor.dump_json(json_path) .

dump_json(*path*)
Writes on file specified by path the JSON representation of titles and subtitles extracted.

Dumps the titles and subtitles according to the hierarchy verified on the document.

The outputfile should be specified and will be suffixed with the ".json" if it's not.

Parameters

- **path** – string containing path to .json file where the dump will
- **not.** (*be done. Its suffixed with ".json" if it's*) –

property json
All titles with its subtitles associated.

All subtitles under the same title are at the same level. Deprecated. Better use *titles_subtitles* or *titles_subtitles_hierarchy*.

reset()
Sets cache to False and reset others internal attributes. Use when for some reason the internal state was somehow modified by user.

property subtitles
All subtitles extracted from the file speficied by self._path.

Returns List[TextTypeBboxPageTuple] each of which having its type attribute equals _TYPE_SUBTITLE

property titles
All titles extracted from the file speficied by self._path.

Returns List[TextTypeBboxPageTuple] each of which having its type attribute equals _TYPE_TITLE

property titles_subtitles
A list with titles and subtitles, sorted according to its reading order.

property titles_subtitles_hierarchy: TitlesSubtitles(titles=<class 'str'>, subtitles=typing.List[str])
All titles and subtitles extracted from the file specified by self._path, hierarchically organized.

Returns the titles and its respectively subtitles

Return type List[*TitlesSubtitles*(str, List[str])]

class dodfminer.extract.pure.utils.title_extractor.TextTypeBboxPageTuple(*text*, *type*, *bbox*, *page*)

property bbox
Alias for field number 2

property page
Alias for field number 3

property text
Alias for field number 0

property type
Alias for field number 1

class dodfminer.extract.pure.utils.title_extractor.**TitlesSubtitles**(*titles*, *subtitles*)

property subtitles
Alias for field number 1

property titles
Alias for field number 0

dodfminer.extract.pure.utils.title_extractor.**extract_titles_subtitles**(*path*)
Extracts titles and subtitles from DODF pdf.

Parameters **path** – str indicating the path for the pdf to have its content extracted.

Returns List[TextTypeBboxPageTuple] containing all titles ans subtitles.

dodfminer.extract.pure.utils.title_extractor.**gen_hierarchy_base**(*dir_path='.'*, *folder='hierarchy'*, *indent=4*, *forced=False*)

Generates json base from all PDFs immediately under dir_path directory. The hiearchy files are generated under dir_path directory. Args:

dir_path: path so folder containing PDFs base_name: titles' base file name forced: proceed even if folder *base_name* already exists indent: how many spaces used will be used for indent

Returns: List[Dict[str, List[Dict[str, List[Dict[str, str]]]]]] e.g: [

{ "22012019": [

{ "PODER EXECUTIVO": []

}, {

"SECRETARIA DE ESTADO DE FAZENDA,

PLANEJAMENTO, ORÇAMENTO E GESTÃO": [

{ "SUBSECRETARIA DA RECEITA": ""

}

]

}

}

] In case of error trying to create *base_name* folder, returns None.

dodfminer.extract.pure.utils.title_extractor.**gen_title_base**(*dir_path='.'*, *base_name='titles'*, *indent=4*, *forced=False*)

Generates titles base from all PDFs immediately under dir_path directory. The base is generated under dir_path directory. :param dir_path: path so base_name will contain all titles

from PDFs under dir_path

Parameters

- **base_name** – titles' base file name
- **indent** – how many spaces used will be used for indent

Returns

dict containing "titles" as key and a list of titles, the same stored at base_name[.json]

dodfminer.extract.pure.utils.title_extractor.**group_by_column**(*elements*, *width*)

Groups elements by its culumns. The sorting assumes they are on the same page and on a 2-column layout.

Essentially a "groupby" where the key is the page number of each span.

Parameters **elements** – Iterable[TextTypeBboxPageTuple] sorted by its page number to be grouped.

Returns A dict with spans of each page, being keys the page numbers.

dodfminer.extract.pure.utils.title_extractor.**group_by_page**(*elements*)

Groups elements by page number.

Essentially a "groupby" where the key is the page number of each span.

Parameters **elements** – Iterable[TextTypeBboxPageTuple] sorted by its page number to be grouped.

Returns A dict with spans of each page, being keys the page numbers.

dodfminer.extract.pure.utils.title_extractor.**invert_text_type_bbox_page_tuple**(*text_type_bbox_page_tuple*)

Reverses the type between _TYPE_TITLE and _TYPE_SUBTITLE.

Parameters **textTypeBboxPageTuple** – instance of TextTypeBboxPageTuple.

Returns copy of textTypeBboxPageTuple with its type field reversed.

dodfminer.extract.pure.utils.title_extractor.**load_blocks_list**(*path*)

Loads list of blocks list from the file specified.

Parameters **path** – string with path to DODF pdf file

Returns A list with page blocks, each element being a list with its according page blocks.

dodfminer.extract.pure.utils.title_extractor.**sort_2column**(*elements*, *width_lis*)

Sorts TextTypeBboxPageTuple iterable.

Sorts sequence of TextTypeBboxPageTuple objects, assuming a full 2-columns layout over them.

Parameters **elements** – Iterable[TextTypeBboxPageTuple]

Returns dictionary mapping page number to its elements sorted by column (assumig there are always 2 columns per page)

dodfminer.extract.pure.utils.title_extractor.**sort_by_column**(*elements*, *width*)

Sorts list elements by columns.

Parameters

- **elements** – Iterable[TextTypeBboxPageTuple].
- **width** – the page width (the context in which all list elements were originally).

Returns

List[TextTypeBboxPageTuple] containing the list elements sorted according to:

1. columns
2. position on column

Assumes a 2-column page layout. All elements on the left column will be placed first of any element on the right one. Inside each columns, reading order is expected to be kept.

dodfminer.extract.pure.utils.title_extractor.**load_blocks_list**(*path*)
Loads list of blocks list from the file specified.

Parameters **path** – string with path to DODF pdf file

Returns A list with page blocks, each element being a list with its according page blocks.

dodfminer.extract.pure.utils.title_extractor.**group_by_column**(*elements*, *width*)
Groups elements by its culumns. The sorting assumes they are on the same page and on a 2-column layout.

Essentially a "groupby" where the key is the page number of each span.

Parameters **elements** – Iterable[TextTypeBboxPageTuple] sorted by its page number to be grouped.

Returns A dict with spans of each page, being keys the page numbers.

dodfminer.extract.pure.utils.title_extractor.**group_by_page**(*elements*)
Groups elements by page number.

Essentially a "groupby" where the key is the page number of each span.

Parameters **elements** – Iterable[TextTypeBboxPageTuple] sorted by its page number to be grouped.

Returns A dict with spans of each page, being keys the page numbers.

dodfminer.extract.pure.utils.title_extractor.**sort_by_column**(*elements*, *width*)
Sorts list elements by columns.

Parameters

- **elements** – Iterable[TextTypeBboxPageTuple].
- **width** – the page width (the context in which all list elements were originally).

Returns

List[TextTypeBboxPageTuple] containing the list elements sorted according to:

1. columns
2. position on column

Assumes a 2-column page layout. All elements on the left column will be placed first of any element on the right one. Inside each columns, reading order is expected to be kept.

dodfminer.extract.pure.utils.title_extractor.**_extract_bold_upper_page**(*page*)
Extracts page content which have bold font and are uppercase.

Parameters **page** – fitz.fitz.Page object to have its bold content extracted.

Returns A list containing all bold (and simultaneously upper) content at the page.

dodfminer.extract.pure.utils.title_extractor.**_extract_bold_upper_pdf**(*doc*)
Extracts bold content from DODF pdf.

Parameters **doc** – DODF pdf file returned by *fitz.open*

Returns a list of list of bold span text

dodfminer.extract.pure.utils.title_extractor.**sort_2column**(*elements*, *width_lis*)
Sorts TextTypeBboxPageTuple iterable.

Sorts sequence of TextTypeBboxPageTuple objects, assuming a full 2-columns layout over them.

Parameters **elements** – Iterable[TextTypeBboxPageTuple]

Returns dictionary mapping page number to its elements sorted by column (assumig there are always 2 columns per page)

dodfminer.extract.pure.utils.title_extractor.**_get_titles_subtitles**(*elements*, *width_lis*)
Extracts titles and subtitles from list. WARNING: Based on font size and heuristic.

Parameters **titles_subtitles** – a list of dict all of them having the keys: size -> float text -> str bbox -> Box page -> int

Returns TitlesSubtitles[List[TextTypeBboxPageTuple], List[TextTypeBboxPageTuple]].

dodfminer.extract.pure.utils.title_extractor.**_get_titles_subtitles_smart**(*doc*, *width_lis*)
Extracts titles and subtitles. Makes use of heuristics.

Wraps _get_titles_subtitles, removing most of impurity (spans not which aren't titles/subtutles).

Parameters **doc** – DODF pdf file returned by *fitz.open*

Returns

TitlesSubtitles(List[TextTypeBboxPageTuple], List[TextTypeBboxPageTuple]).

dodfminer.extract.pure.utils.title_extractor.**extract_titles_subtitles**(*path*)
Extracts titles and subtitles from DODF pdf.

Parameters **path** – str indicating the path for the pdf to have its content extracted.

Returns List[TextTypeBboxPageTuple] containing all titles ans subtitles.

class dodfminer.extract.pure.utils.title_extractor.**ExtractorTitleSubtitle**(*path*)
Use this class like that: >> path = "path_to_pdf" >> extractor = ExtractorTitleSubtitle(path) >> # To extract titles >> titles = extractor.titles >> # To extract subtitles >> titles = extractor.subtitles >> # To dump titles and subtitles on a json file >> json_path = "valid_file_name" >> extractor.dump_json(json_path) .

dump_json(*path*)
Writes on file specified by path the JSON representation of titles and subtitles extracted.

Dumps the titles and subtitles according to the hierarchy verified on the document.

The outputfile should be specified and will be suffixed with the ".json" if it's not.

Parameters

- **path** – string containing path to .json file where the dump will
- **not.** (*be done. Its suffixed with ".json" if it's*) –

property json
All titles with its subtitles associated.

All subtitles under the same title are at the same level. Deprecated. Better use *titles_subtitles* or *titles_subtitles_hierarchy*.

reset()
Sets cache to False and reset others internal attributes. Use when for some reason the internal state was somehow modified by user.

property subtitles
All subtitles extracted from the file speficied by self._path.

Returns List[TextTypeBboxPageTuple] each of which having its type attribute equals _TYPE_SUBTITLE

property titles
All titles extracted from the file speficied by self._path.

Returns List[TextTypeBboxPageTuple] each of which having its type attribute equals _TYPE_TITLE

property titles_subtitles
A list with titles and subtitles, sorted according to its reading order.

property titles_subtitles_hierarchy: TitlesSubtitles(titles=<class 'str'>, subtitles=typing.List[str])
All titles and subtitles extracted from the file specified by self._path, hierarchically organized.

Returns the titles and its respectively subtitles

Return type List[*TitlesSubtitles*(str, List[str])]

dodfminer.extract.pure.utils.title_extractor.**gen_title_base**(*dir_path='.'*, *base_name='titles'*, *indent=4*, *forced=False*)
Generates titles base from all PDFs immediately under dir_path directory. The base is generated under dir_path directory. :param dir_path: path so base_name will contain all titles

from PDFs under dir_path

Parameters

- **base_name** – titles' base file name
- **indent** – how many spaces used will be used for indent

Returns

dict containing "titles" as key and a list of titles, the same stored at base_name[.json]

dodfminer.extract.pure.utils.title_extractor.**gen_hierarchy_base**(*dir_path='.'*, *folder='hierarchy'*, *indent=4*, *forced=False*)

Generates json base from all PDFs immediately under dir_path directory. The hiearchy files are generated under dir_path directory. Args:

dir_path: path so folder containing PDFs base_name: titles' base file name forced: proceed even if folder *base_name* already exists indent: how many spaces used will be used for indent

Returns: List[Dict[str, List[Dict[str, List[Dict[str, str]]]]]] e.g: [

{ "22012019": [

{ "PODER EXECUTIVO": []

}, {

"SECRETARIA DE ESTADO DE FAZENDA,

PLANEJAMENTO, ORÇAMENTO E GESTÃO": [

{ "SUBSECRETARIA DA RECEITA": ""

}

]

}

}

] In case of error trying to create *base_name* folder, returns None.

CHAPTER

TEN

POLISHED CORE

Table of Contents

10.1 The Act Extractor Class

10.1.1 Returning Objects

The methods in this section return objects or vectors of objects.

10.1.2 Returning Dataframes

The methods in this section return dataframes or vectors of dataframes.

CHAPTER

ELEVEN

POLISHED HELPER

CHAPTER

TWELVE

ACTS

Table of Contents

Acts are always built as a child class from the Base class `Atos`. Following are the base class structure and a guide for implementating your own act. Also, a list of implementated and missing acts are presented.

12.1 Base Class

12.2 Implementing new acts

The Acts base class is build in a way to make easy implementation of new acts. A programmer seeking to help in the development of new acts, need not to worry about anything, besides the regex or ner itself.

Mainly, the following funcions need to be overwrited in the child class.

12.2.1 Regex Methods

In case you want to extract through regex, the following funcions needs to be written:

`ActRegex._rule_for_inst()`
Rule for extraction of the act

> **Warning:** Must return a regex rule that finds an act in two parts, containing a head and a body. Where only the body will be used to search for proprieties.

Raises **NotImplementedError** – Child class needs to overwrite this method.

ActRegex.**_prop_rules**()
Rules for extraction of the proprieties.

Must return a dictionary of regex rules, where the key is the propriety type and the value is the rule.

Raises **NotImplementedError** – Child class needs to overwrite this method

Additionaly, if the programmer whishes to change the regex flags for his/her class, they can overwrite the following function in the child class:

classmethod ActRegex.**_regex_flags**()
Flag of the regex search

12.2.2 NER Methods

If NER will be used, you shall add a trained model to the acts/models folder. Also the following method should be overwrited in your act:

12.2.3 Change the Core File

After all functions have been implemented, the programmer needs to do a minor change in the core file. The following must be added:

```
from dodfminer.extract.polished.acts.act_file_name import NewActClass
_acts_ids["new_act_name"] = NewActClass
```

12.3 Base Class Mechanisms

One does not access directly none of those functions, but they are listed here in case the programmer implementing the act needs more informations.

12.4 Implemented Acts

- Abono
- Aposentadoria
- Exoneração
- Nomeação
- Retificações
- Reversões
- Substituições
- Cessões
- Tornar sem efeito Aposentadoria
- Exoneração de Cargos Efetivos

CHAPTER

THIRTEEN

REGEX BACKEND

Regex backend for act and propriety extraction.

This module contains the ActRegex class, which have all that is necessary to extract an act and, its proprieties, using regex rules.

class dodfminer.extract.polished.backend.regex.**ActRegex**

Act Regex Class.

This class encapsulate all functions, and attributes related to the process of regex extraction.

Note: This class is one of the fathers of the Base act class.

_flags

All the regex flags which will be used in extraction.

_rules

The regex rules for proprieties extraction.

_inst_rule

The regex rule for act extraction.

_find_prop_value(*rule*, *act*)

Find a single proprietie in an single act.

Parameters

- **rule** (*str*) – The regex rule to search for.
- **act** (*str*) – The act to apply the rule.

Returns The found propriety, or a nan in case nothing is found.

_prop_rules()

Rules for extraction of the proprieties.

Must return a dictionary of regex rules, where the key is the propriety type and the value is the rule.

Raises **NotImplementedError** – Child class needs to overwrite this method

classmethod _regex_flags()

Flag of the regex search

_regex_instances()

Search for all instances of the act using the defined rule.

Returns List of all act instances in the text.

_regex_props(*act_raw*)
Create an act dict with all its proprieties.

Parameters **act_raw** (*str*) – The raw text of a single act.

Returns The act, and its props in a dictionary format.

_rule_for_inst()
Rule for extraction of the act

> **Warning:** Must return a regex rule that finds an act in two parts, containing a head and a body. Where only the body will be used to search for proprieties.

Raises **NotImplementedError** – Child class needs to overwrite this method.

CHAPTER

FOURTEEN

NER BACKEND

CHAPTER

FIFTEEN

ACKNOWLEDGEMENTS

We gratefully acknowledge the contributions of the many people who helped get this project off of the ground, including people who beta tested the software, gave feedback on the material, improved dependencies of DODFMiner code in service of this release, or otherwise supported the project. Given the number of people who were involved at various points, this list of names may not be exhaustive. (If you think you should have been listed here, please do not hesitate to reach out.)

In no particular order, thank you Khalil Carsten, Renato Nobre, Isaque Alves, Leonardo Maffei, João Zarbiélli, Felipe Almeida, Davi Alves, Fabrício Braz, Thiago Faleiros and Nilton Correia.

We are also grateful to the University of Brasília, TCDF and Finatec (Fundação de Empreendimentos Científicos e Tecnológicos for the partnership, and the FAPDF (Fundação de Apoio à Pesquisa do Distrito Federal) for the funding.

CHAPTER

SIXTEEN

ABOUT THE KNEDLE TEAM

The project “KnEDLe - Knowledge Extraction from Documents of Legal content” is a partnership among FAPDF (Fundação de Apoio à Pesquisa do Distrito Federal), UnB (the University of Brasília) and Finatec (Fundação de Empreendimentos Científicos e Tecnológicos), sponsored by FAPDF. This project was proposed in order to employ official publications as a research object and to extract knowledge. The objective is to develop intelligent tools for extracting structured information from such publications, aiming to facilitate the search and retrieval of information, increasing government transparency and facilitating audit tasks and detecting problems related to the use of public resources.

16.1 Check our website

PYTHON MODULE INDEX

d

INDEX

Symbols

A

B

S

T

X

Y