

extract_sizes.py

MIT License

Copyright 2023 auto_anki

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

File completing step 2: given a pdf document return a dictionary of headers and paragraphs

Given a filename,
opens the PDF and extracts words & metadata from each slide

:param file: String representing file path
:type: string
:rtype: dict
:return: dict, Contains extracted metadata & words from all slides

iterate through the text blocks

block contains text

iterate through the text lines

iterate through the text spans

Helper function to get unique sizes within a PDF

:param doc: The list of blocks within a PDF
:type: list
:rtype: list
:return: a list of unique font sizes

ensuring object is not None/empty

for each page in our document

```
import re
import fitz

def extract_words(file: str) -> dict:

    document = fitz.open(file, filetype="pdf")
    doc_data = {}
    doc_data["meta_data"] = document.metadata
    doc_data["data"] = []
    for index, page in enumerate(document):
        page_data = {}
        page_data["slide"] = index+1
        page_data["blocks"] = []
        blocks = page.get_text("dict")["blocks"]

        for block in blocks:

            if block['type'] == 0:

                for line in block["lines"]:

                    for span in line["spans"]:
                        page_data["blocks"].append({
                            "text": re.sub(r"\W{3,}", " ", span["text"]),
                            "size": span["size"]
                        })
                    doc_data["data"].append(page_data)

    return doc_data

def get_sizes(doc: dict) -> list:

    if not doc:
        return []

    unique_fonts = set()

    for page in doc['data']:
```

get the individual text blocks

can also get font and color

sort the fonts for later filtering

Categorizes each text into L, M, or S.

```
:param unique_fonts: a list of unique fonts in the powerpoint
:type unique_fonts: list
:param doc: a list of blocks per each document page
:type doc: dict
:rtype: list
:return: a list of dicts categorizing texts into respective categories
```

check that both are not None, or empty

The Header will be the top 2 font sizes top font size is Title, second would be header

get the individual text blocks

if the text size is smaller than header or title

trim any extra whitespace

Given a pdf document,
Returns a dictionary of Headers, Paragraphs, and page number

```
:param doc: a PDF document containing only words
:type: dict
:rtype: list
:return: dict categorizing each text into its respective category
```

```
for block in page['blocks']:

    unique_fonts.add(round(block['size']))

sorted_fonts = sorted(list(unique_fonts))
return sorted_fonts

def tag_text(unique_fonts: list, doc: dict) -> list:

    if not doc or not unique_fonts:
        return []

    header_lim = unique_fonts[-2]
    all_pages = []

    for page in doc['data']:
        text_dict = {'Header': "", 'Paragraph': "", 'slide': page['slide']}

        for block in page['blocks']:

            if block['size'] < header_lim:
                text_dict['Paragraph'] += block['text'] + " "
            else:
                text_dict['Header'] += block['text'] + " "

        text_dict['Paragraph'] = text_dict['Paragraph'].strip()
        text_dict['Header'] = text_dict['Header'].strip()
        all_pages.append(text_dict)
    return all_pages

def text_to_groupings(doc: dict) -> list:

    font_count = get_sizes(doc)
    lst_fonts = tag_text(font_count, doc)
    return lst_fonts
```