

wordprocessing.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. wordprocessing.py

```
import string
from collections import OrderedDict, Counter
import sys
import re
import spacy
import numpy as np
```

```
def keyword_extractor(data: list) -> list:
```

Function to extract keywords from the headers and paragraphs of slides

:param data: The list of dictionaries of the form
:type: [{"Header":"","Paragraph":"","slide:int}]
:return: The list of dictionaries with keywords extracted of the form
:rtype: [{"Header":"","Paragraph":"","Header_keywords": [],
"Paragraph_keywords": [],
slide:int}]

```
try:
    nlp = spacy.load("en_core_web_lg")
except OSError:
    sys.exit()
pos_tag = ["NOUN"]
dep_tag = ["nsubj"]
for slide in data:
    doc_header = nlp(slide["Header"].lower())
    doc_paragraph = nlp(slide["Paragraph"].lower())
    header_keywords = []
    paragraph_keywords = []
    for token in doc_header:
        if token.text in nlp.Defaults.stop_words or token.is_punct:
            continue
        if token.pos_ in pos_tag or token.dep_ in dep_tag:
            word = re.sub(r"[^0-9a-zA-Z]+", " ", token.text)
            word = word.strip()
            if len(word) >= 3:
                header_keywords.append(word)
    for token in doc_paragraph:
        if token.text in nlp.Defaults.stop_words or token.is_punct:
            continue
        if token.pos_ in pos_tag or token.dep_ in dep_tag:
            word = re.sub(r"[^a-zA-Z]+", " ", token.text)
            word = word.strip()
            if len(word) >= 3:
                paragraph_keywords.append(word)
    slide["Header_keywords"] = header_keywords
    slide["Paragraph_keywords"] = paragraph_keywords
return data
```

```
def duplicate_word_removal(data: list) -> list:
```

Removes duplicate words

:param data: The list of dictionaries of the form
:type: [{"Header":"","Header_keywords": [], "Paragraph_keywords": [], slides:[int]}]
:return: The list of dictionaries with duplicate keywords removed of the form
:rtype: [{"Header":"","Header_keywords": [], "Paragraph_keywords": [], slides:[int]}]

```
for dictionary in data:
    ordered_headers = list(OrderedDict.fromkeys(
        dictionary['Header_keywords']))
    dictionary['Header_keywords'] = ordered_headers

    ordered_paragraph = list(OrderedDict.fromkeys(
        dictionary['Paragraph_keywords']))
    dictionary['Paragraph_keywords'] = ordered_paragraph
return data
```

Function to merge slides with the same header.

```
:param data: The list of dictionaries of the form
:type: [{"Header":"","
        "Paragraph":"","
        "Header_keywords": [],
        "Paragraph_keywords": [],
        slide:int}]
:return: The list of dictionaries where slides containing the same header are merged
:rtype: [{"Header":""," "Header_keywords": [], "Paragraph_keywords": [], slides:[int]}]
```

```
def merge_slide_with_same_headers(data: list) -> list:
```

```
merged = []
headers = []
for slide in data:
    if slide["Header"] not in headers:
        headers.append(slide["Header"])
        paragraph_keywords = []
        slide_numbers = []
        for data_1 in [data_2 for data_2 in data if data_2["Header"] == slide["Header"]]:
            paragraph_keywords += data_1["Paragraph_keywords"]
            slide_numbers.append(data_1["slide"])
        merged.append({"Header": slide["Header"], "Header_keywords": slide["Header_keywords"],
                      "Paragraph_keywords": paragraph_keywords, "slides": slide_numbers})

return merged
```

Function to merge slides with the same slide number into a single one.

```
:param data: The list of dictionaries of the form
:type: [{"Header":"","
        "Paragraph":"","
        "Header_keywords": [],
        "Paragraph_keywords": [],
        slide:int}]
:return: The list of dictionaries where slides containing the same slide number are merged
:rtype: [{"Header":""," "Header_keywords": [], "Paragraph_keywords": [], slide:int}]
```

```
def merge_slide_with_same_slide_number(data: list) -> list:
```

```
merged = []
slide_number = []
for slide in data:
    if slide["slide"] not in slide_number:
        slide_number.append(slide["slide"])
        header_keywords = []
        paragraph_keywords = []
        for data_1 in [data_2 for data_2 in data if data_2["slide"] == slide["slide"]]:
            header_keywords += data_1["Header_keywords"]
            paragraph_keywords += data_1["Paragraph_keywords"]
        merged.append({"Header": slide["Header"], "Header_keywords": header_keywords,
                      "Paragraph_keywords": paragraph_keywords,
                      "slide": slide["slide"]})

return merged
```

Constructs a search query given a PDF data

```
:param data: The list of data
:type: list
:return: List of words to search
:rtype: list
```

```
def construct_search_query(data: list) -> list:
```

```
header_keywords = []
paragraph_keywords = []
for item in data:
    header_keywords += item["Header_keywords"] * len(item["slides"])
    paragraph_keywords += item["Paragraph_keywords"] * len(item["slides"])
header_counts = Counter(header_keywords)
paragraph_counts = Counter(paragraph_keywords)
header_mean = np.array(list(header_counts.values())).mean()
paragraph_mean = np.array(list(paragraph_counts.values())).mean()
header_search = []
paragraph_search = []
for key, value in header_counts.items():
    if value > header_mean:
        header_search.append(key)
for key, value in paragraph_counts.items():
    if value > paragraph_mean:
        paragraph_search.append(key)
return header_search + paragraph_search
```

Extracts nouns using Spacy

```
:param data: list of PDF data
:type: list
:return: list of data with nouns extracted
:rtype: list
```

```
def extract_noun_chunks(data: list) -> list:
```

```
try:
    nlp = spacy.load("en_core_web_lg")
except OSError:
    sys.exit()
for slide in data:
    doc_header_noun_chunks = nlp(slide["Header"].lower()).noun_chunks
    doc_paragraph_noun_chunks = nlp(slide["Paragraph"].lower()).noun_chunks
    header_keywords = []
    paragraph_keywords = []
    for token in doc_header_noun_chunks:
        processed_words = []
        words = token.text.split()
        for word in words:
            word = re.sub(r"^[a-zA-Z]+", "", word).strip()
```

```
        if word in nlp.Defaults.stop_words or word in string.punctuation:
            continue
        if len(word) >= 3:
            processed_words.append(word)
        if len(processed_words) >= 2:
            header_keywords.append(" ".join(processed_words))
    for token in doc_paragraph_noun_chunks:
        processed_words = []
        words = token.text.split()
        for word in words:
            word = re.sub(r"^[a-zA-Z]+", "", word).strip()
            if word in nlp.Defaults.stop_words or word in string.punctuation:
                continue
            if len(word) >= 3:
                processed_words.append(word)
        if len(processed_words) >= 2:
            paragraph_keywords.append(" ".join(processed_words))
    slide["Header_keywords"] = header_keywords
    slide["Paragraph_keywords"] = paragraph_keywords
return data
```