MaOW - Map Of Words

Spis Treści

Specyfikacja wymagań

Opis ogólny:

Program ma na celu przetwarzanie tekstów zapisanych w plikach PDF. Program używa do odczytu zawartości plików PDF biblioteki PyPDF2 . Następnie, przy wykorzystaniu biblioteki SpaCy do analizy języka naturalnego (NLP), program znajduje lematy słów, określa części mowy oraz usuwa stopwords. Na podstawie przetworzonych danych tekstowych tworzy mapy i chmury słów .

Wymagania funkcjonalne:

- Program umożliwia użytkownikowi podanie ścieżki do plików PDF, które mają zostać przetworzone.
- Program odczytuje zawartość plików PDF za pomocą biblioteki PyPDF2.
- Tekst odczytany z plików PDF jest przetwarzany przy użyciu biblioteki SpaCy w celu przetworzenia tekstu wykonać następujące zadania:
  - Wykonanie tokenizacji.
  - Znalezienie lematów słów w tekście.
  - Określenie części mowy słowa.
  - Usunięcie stopwords (słów bez znaczenia, np. "a", "the", "in") z tekstu.
- Program generuje mapy i chmury słów na podstawie przetworzonych danych tekstowych przy użyciu bibliotek: WordCloud i Matplotlib.
- Wyniki generacji map i chmur słów są wyświetlane i zapisywane w odpowiednich formatach (np. obraz JPG).

Wymagania niefunkcjonalne:

- Program powinien być napisany w języku Python i wykorzystywać wersję Pythona zgodną z bibliotekami SpaCy, WordCloud i PyPDF2.
- Program powinien obsługiwać pliki PDF w formacie tekstowym (niezabezpieczone hasłem).
- Program powinien być wydajny i umożliwiać przetwarzanie większych plików PDF w rozsądnym czasie.

Wymagania dotyczące interfejsu użytkownika:

Program jest wykorzystywany przy użyciu wiersza poleceń i umożliwia:

- podanie ścieżki do plików PDF do przetworzenia.
- wyboru języka pod przetwarzanie tekstu z dostępnych języków.
- wybranie rodzaju wykresu:
  - Wordcloud - pojedyńcza chmura słów.
  - section_clouds - obraz chmur słów dla poszczególnych sekcji, ukazujących rozkład słów w dokumencie
  - Wordmap - wykres postaci mapy słów (mapy myśli), ukazujący rozkład słów w dokumencie oraz częstość ich występowania w dokumencie.
- podanie ścieżki oraz nazwy zapisywanego obrazu.

Metody

program

- **pdf_path** - ścieżka pliku PDF.
- ***language*** - język, w którym jest napisany nasz dokument.
  language= '*english*' - domyślnie ustawiony język angielski;
  dostępne języki: '*english*' (angielski).
- ***plot*** - zwracany rodzaj wykresu.
  plot='Wordcloud' - domyślnie ustawiony wykres Wordcloud;
  dostępne wykresy: *'Wordcloud', 'section_clouds', 'Wordmap'*
- ***filename*** - nazwa zapisywanego wykresu jako obraz JPG.
  filename='' - domyślnie pusty string, skutkuje nie zapisaniem pliku na komputerze.
- ***path*** - ścieżka do miejsca, w którym ma zostać zapisywany wykres
  path='' - domyślnie pusty string, skutkuje zapisaniem pliku w obecnym katalogu.
- ***width***, ***height*** - szerokość i wysokość wykresu (dotyczy tylko '*Wordcloud' i 'section_clouds'*).
  obie zmienne ustawione domyślnie na 500
- ***background*** - tło wykresu (dotyczy tylko '*Wordcloud' i 'section_clouds'*)
  domyślnie ustawione białe (*"white"*)
- ***colormap*** - mapa kolorów (dotyczy tylko '*Wordcloud*')
  domyślnie ustawiona (*'magma')*

```
def program(pdf_path, language='english', plot='Wordcloud', filename='', path='',
               width = 500, height = 500, background = "white",
               colormap = "magma"):

    languages=['english']
    if language.lower() not in languages:
        language='english'
        print("Language not avaible. Language set to English.")

    information, number_of_pages = extract_information(pdf_path)
    section_list=[]
    section = [0, round(number_of_pages/4),round(number_of_pages/4)*2,
              round(number_of_pages/4)*3,number_of_pages]

    for i in range(1,5):
        word_list=[]
        for page in range(section[i-1], section[i]):
            text = read_page(pdf_path, page)
            word_list.extend(text_preparation(text, language))
        section_list.append(word_list)

    word_list = [word for section in section_list for word in section_list][0]

    if plot=='Wordcloud':
        wordCloud(word_list, information, filename, path,
                  width, height, background, colormap)

    elif plot=='section_clouds':
        wordClouds(section_list, information, filename, path,
                    width, height, background)

    elif plot=='Wordmap':
        wordMap(section_list, information, filename, path)
```

extract_information

Funkcja zwracająca informacje *information* (zawiera informacje m.in o autorze tytule) oraz liczbę stron *number_of_pages.*

- **pdf_path** - ścieżka pliku PDF.

```
def extract_information(pdf_path):
    from PyPDF2 import PdfReader
    with open(pdf_path, 'rb') as f:
        pdf = PdfReader(f)
        information = pdf.metadata
        number_of_pages = len(pdf.pages)

    return information, number_of_pages
```

▼ **Example**

```
extract_information("The Stranger - Albert Camus.pdf")

Out[1]:
({'/Author': 'Albert Camus',
  '/CreationDate': "D:20130622043106-04'00'",
  '/Creator': 'calibre (0.9.34) [http://calibre-ebook.com]',
  '/Keywords': 'Classics, Fiction, ST, CS',
  '/ModDate': "D:20130622215635-05'00'",
  '/Producer': 'Adobe Acrobat 10.1.7 Paper Capture Plug-in with ClearScan',
  '/Subject': 'Fiction, Classics',
  '/Title': 'The Stranger'},
 137)
```

read_page

Funkcja zwracająca tekst z konkretnej strony dokumentu PDF.

- **pdf_path** - ścieżka pliku PDF.
- **page** - numer strony.

```
def read_page(pdf_path, page):
    from PyPDF2 import PdfReader
    reader = PdfReader(pdf_path)
    page = reader.pages[page]

    return page.extract_text()
```

text_preparation

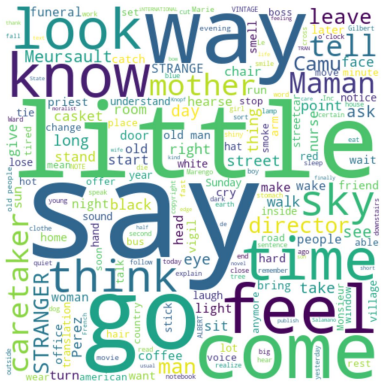
Funkcja zwracająca listę słów po przetworzeniu tekstu.

- **text** - *tekst w postaci String do przetworzenia.*
- ***language*** - język, dla którego ładujemy słownik słów z biblioteki SpaCy.
  language= 'english' - domyślnie ustawiony język angielski;

```
def text_preparation(text, language='english'):
    import spacy

    if language=='english':
        nlp = spacy.load("en_core_web_sm")
        stopwords = ['ing', 'let', 'away']

    text = nlp(text)

    unwanted_pos = ["SYM", "NUM", "PUNCT", "INTJ", "AUX"]
    words=[]
    for token in text:
        if len(token)>1:
            if token.pos_ not in unwanted_pos and not token.is_stop:
                if not (len(token)==2 and token.lemma_[-1]=='.'):
                    words.append(token.lemma_)

    words = [w for w in words if w.lower() not in stopwords]

    return words
```

Wykresy

wordCloud



Chmura słów stworzona na podstawie książki A.Camus "The Stranger"

Chmura słów stworzona na podstawie artykułu *pyResearchInsights*—An open-source Python package for scientific text analysis

- **word_list** - lista, zawierająca słowa.
- ***filename*** - nazwa zapisywanego wykresu jako obraz JPG.
  filename='' - domyślnie pusty string, skutkuje nie zapisaniem pliku na komputerze.
- ***path*** - ścieżka do miejsca, w którym ma zostać zapisywany wykres.
  path='' - domyślnie pusty string, skutkuje zapisaniem pliku w obecnym katalogu.
- ***width***, ***height*** - szerokość i wysokość wykresu (dotyczy tylko '*Wordcloud' i 'section_clouds'*).
  obie zmienne ustawione domyślnie na 500 .
- ***background*** - tło wykresu. (dotyczy tylko '*Wordcloud' i 'section_clouds'*)
  domyślnie ustawione białe (*"white"*) .
- ***colormap*** - mapa kolorów. (dotyczy tylko '*Wordcloud'*)
  domyślnie ustawiona. (*'magma')*

```
def wordCloud(word_list,  filename='', path='', width = 500, height = 300, background = "white",
              colormap = "magma"):
    import matplotlib.pyplot as plt
    from wordcloud import WordCloud
    import os

    wordcloud = WordCloud(width = width, height = height,
                colormap = colormap,
                 background_color=background).generate(" ".join(word_list))

    page, ax = plt.subplots(nrows = 1, ncols = 1, dpi = 300,
                                  constrained_layout = True)

    ax.imshow(wordcloud, interpolation='bilinear')
    ax.axis("off")
    plt.show()

    if filename!='':
        if path!='':
            if not os.path.isdir(path):
                os.makedirs(path)
            page.savefig('{path}/{name}_wordcloud.jpg'.format(path = path, name = filename))

        else:
            page.savefig('{name}_wordcloud.jpg'.format(name = filename))
```

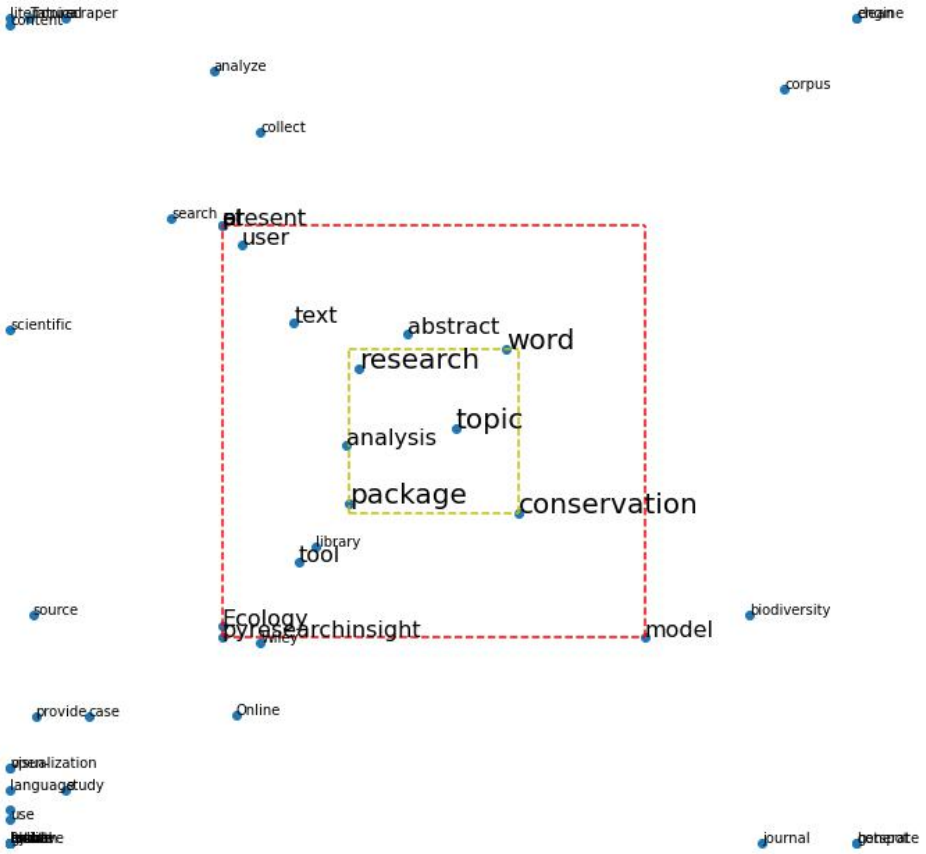
section_clouds

Chmury słów (klastry) stworzone na podstawie książki A.Camus “The Stranger”

- **section_list** - lista, zawierająca listę słów dla każdej z sekcji.
  (pierwsza ćwierć książki, druga ćwierć książki, trzecia ćwierć książki, czwarta ćwierć książki)
- ***filename*** - nazwa zapisywanego wykresu jako obraz JPG.
  filename='' - domyślnie pusty string, skutkuje nie zapisaniem pliku na komputerze.
- ***path*** - ścieżka do miejsca, w którym ma zostać zapisywany wykres.
  path='' - domyślnie pusty string, skutkuje zapisaniem pliku w obecnym katalogu.
- ***width***, ***height*** - szerokość i wysokość wykresu. (dotyczy tylko ‘*Wordcloud’ i ‘section_clouds’*).
  obie zmienne ustawione domyślnie na 500 .
- ***background*** - tło wykresu. (dotyczy tylko ‘*Wordcloud’ i ‘section_clouds’*)
  domyślnie ustawione białe (*"white"*) .

```
def wordClouds(section_list, filename='', path='', width = 500, height = 300, background = "white"):
    import matplotlib.pyplot as plt
    from wordcloud import WordCloud
    import os

    clusters = [[],[],[],[],[],[],[],[],[]]

    for i in range(0,4):
        for word in section_list[i]:
            if word in section_list[(i+2)%4]:
                clusters[8].append(word)
            elif word in section_list[(i+1)%4]:
                if word in section_list[(i+3)%4]:
                    clusters[8].append(word)
                else:
                    clusters[i + 4].append(word)
            elif word in section_list[(i+3)%4]:
                clusters[(i+3)%4 + 4].append(word)
            else:
                clusters[i].append(word)

    order=[0,4,1,7,8,5,3,6,2]
    colormaps=["winter", "winter", "winter", "winter", "cool", "cool", "cool","cool", "plasma"]
    wordclouds=[]
    for i in range(9):
        if len(clusters[order[i]])>0:
            wordclouds.append(WordCloud(width = width, height = height,
                  colormap = colormaps[order[i]], background_color=background).generate(" ".join(clusters[orde
        else: wordclouds.append(0)


    page = plt.figure(figsize=(20,15))
    for i in range(9):
        im = page.add_subplot(3, 3, i+1)
        im.axis("off")
        if wordclouds[i]!=0:
            im.imshow(wordclouds[i], interpolation='bilinear')


    plt.show()

    if filename!='':
        if path!='':
            if not os.path.isdir(path):
                os.makedirs(path)

            page.savefig('{path}/{name}_sectionClouds.jpg'.format(path = path, name = filename))
        else:
            page.savefig('{name}_sectionClouds.jpg'.format(name = filename))
```

Wordmap



Chmura słów stworzona na podstawie artykułu *pyResearchInsights—An open-source Python package for scientific text analysis*

- **section_list** - lista, zawierająca listę słów dla każdej z sekcji.
  (pierwsza ćwierć książki, druga ćwierć książki, trzecia ćwierć książki, czwarta ćwierć książki)
- ***filename*** - nazwa zapisywanego wykresu jako obraz JPG.
  filename='' - domyślnie pusty string, skutkuje nie zapisaniem pliku na komputerze.
- ***path*** - ścieżka do miejsca, w którym ma zostać zapisywany wykres.
  path='' - domyślnie pusty string, skutkuje zapisaniem pliku w obecnym katalogu.

```
def wordMap(section_list, filename='', path=''):
    import numpy as np
    import matplotlib.pyplot as plt
    import os

    freq ={}

    for i in range(len(section_list)):
        for word in section_list[i]:
            if word.lower() in freq.keys():
                freq[word.lower()][i]+=1
            elif (word[0].upper()+word[1:] in freq.keys()):
                freq[word[0].upper()+word[1:]][i]+=1
            else:
                freq[word]=[0,0,0,0]
                freq[word][i]+=1

    wspol={}
    fontsize=[]
    max_freq = np.max(map(np.max, freq.values()))

    sum_freq = np.array(list(map(sum, freq.values())))

    directions=[[-1,1],[1,1],[1,-1],[-1,-1]]
    min_words=np.max(sum_freq)*0.15
    mean = np.mean(sum_freq[sum_freq>=min_words])

    for word in freq.keys():
        #grupa startowa
        nr_words=sum(freq[word])
        if nr_words>=min_words:
            step_x=0
            step_y=0
            for i in range(4):
                step_x += directions[i][0]*freq[word][i]/nr_words
                step_y += directions[i][1]*freq[word][i]/nr_words
            if step_x==0:
                step_x=0.0001*directions[freq[word].index(np.max(freq[word]))][0]
            if step_y==0:
                step_y=0.0001*directions[feq[word].index(np.max(freq[word]))][1]

            if sum(freq[word])>=mean:
                if sum(freq[word])>=np.mean(sum_freq[sum_freq>=mean]):
                    lim_up=np.max(sum_freq)
                    lim_low=np.mean(sum_freq[sum_freq>=mean])
                    n=10
                    fs=20

                else:
                    lim_up = np.mean(sum_freq[sum_freq>=mean])
                    lim_low = mean
                    n=25
                    fs=16

            else:
                lim_up=mean
                lim_low =min_words
                n=50
                fs=10

            x=n*step_x
            y=n*step_y

            if step_x>0:
                x+=(lim_up-nr_words)/(lim_up-lim_low)*n
                if x>n:
                    x=n
            elif step_x<0:
                x-=(lim_up-nr_words)/(lim_up-lim_low)*n
                if x<-n:
                    x=-n
```

```
        if step_y>0:
            y+=(lim_up-nr_words)/(lim_up-lim_low)*n
            if y>n:
                y=n
        elif step_y<0:
            y-=(lim_up-nr_words)/(lim_up-lim_low)*n
            if y<-n:
                y=-n

        wspol[word]=[x,y]
        fontsize.append(fs)


x = np.transpose(list(wspol.values()))[0]
y = np.transpose(list(wspol.values()))[1]
n = wspol.keys()

page, ax = plt.subplots(figsize=(12,12))
ax.scatter(x,y)

for i, txt in enumerate(n):
    ax.annotate(txt, (x[i], y[i]), fontsize=fontsize[i])

plt.plot([-10,10],[10,10],'y--')
plt.plot([10,10],[-10,10],'y--')
plt.plot([-10,-10],[-10,10],'y--')
plt.plot([-10,10],[-10,-10],'y--')

plt.plot([-25,25],[25,25],'r--')
plt.plot([25,25],[-25,25],'r--')
plt.plot([-25,-25],[-25,25],'r--')
plt.plot([-25,25],[-25,-25],'r--')

plt.axis('off')
plt.show()


if filename!='':
    if path!='':
        if not os.path.isdir(path):
            os.makedirs(path)

        page.savefig('{path}/{name}_WordMap.jpg'.format(path = path, name = filename))
    else:
        page.savefig('{name}_WordMap.jpg'.format(name = filename))
```