Pontifícia Universidade Católica do Rio de Janeiro

Departamento de Informática

Documentação
Similaridade de documentos com NLP

Autoria:

Vanessa Eufrauzino Pacheco

Orientação:

Sérgio Lifshitz

Sumário

Lista de Figuras

1 Módulos *Python* utilizados

A maioria dos módulos utilizados não se faz necessária a instalaçã, já que é nativa do Anaconda navigator, no environment conda forge, que foi utilizado. Os que necessitaram de instalação são exibidos abaixo.

- !pip install pytest
- !pip install unittest
- !pip install tika - necessita da instalação de java
- !pip install PyPDF2
- !pip install python-docx
- !pip install -U spacy
- !python -m spacy download pt_core_news_sm
- !python -m spacy download pt_core_news_md
- !python -m spacy download pt_core_news_lg

Abaixo são exibidos todos os módulos utilizados bem como a razão da utilização

- import os - acesso ao diretório
- import re - ajuste de codificação
- import unicodedata - ajuste de codificação
- import warnings - manutenção de warnings
- import PyPDF2 - exportação de textos de PDF
- from tika import parser - exportação de textos de PDF
- import nltk - linguagem natural
- import pytest - teste unitário
- import unittest - teste unitário
- import spacy - similaridade entre documentos
- from PyPDF2 import PdfFileWriter, PdfFileReader

- nltk.download(’punkt’)
- nltk.download(’stopwords’)
- from nltk.corpus import stopwords - lista de stopwords
- from nltk import word_tokenize,sent_tokenize - tokenização
- import string - linguagem natural
- from string import punctuation - lista de pontuação
- from docx import Document - extração de texto .docx
- warnings.filterwarnings("ignore")

São utilizados dois módulos de extração de documentos em pdf para ajustes de encode.

2 Software

Nessa sessão será exibida cada função existente no software assim como parâmetros de entrada e resultado esperado na saída.
Na figura 1 é exibido um fluxo do software de maneira simplificada.

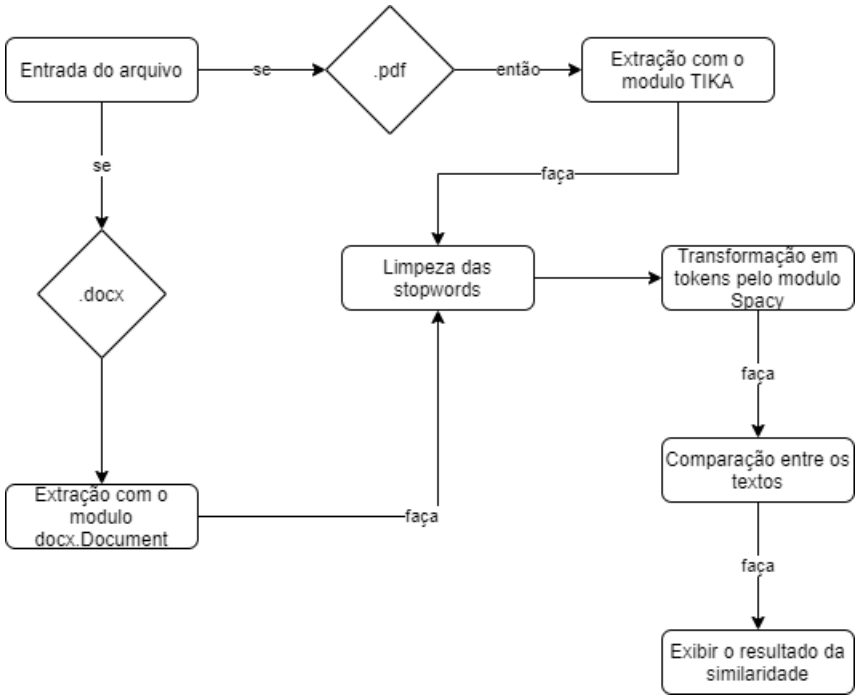

Figura 1: Fluxo do sistema

2.1 Funções

Todas as funções são chamadas de maneira interna, onde só se imputa um parâmetro na ultima função, a de similaridade, que é o diretório onde se encontram os documentos. A partir de então é utilizado o retorno de cada passo para o passo seguinte.

2.1.1 Import_arquivos

A função **import_arquivos**, exibida abaixo, tem como parâmetro de entrada o diretório dos arquivos que se deseja observar a similaridade.

```
def import_arquivos(diretorio):
arquivos = [arq for arq in os.listdir(diretorio)
            if (arq.lower().endswith(".pdf") | arq.lower().endswith(".docx"))]
return(arquivos)
```

Nessa função, os títulos dos documentos são inseridos em uma lista, padronizados em caixa baixa e como retorno tem-se a lista de documentação existentes no diretório com terminações .pdf ou .docx.

2.1.2 Object_create

Na função **object_create** tem como entrada o diretório e a cada documento retornado na função import_arquivos, é verificado se a extensão é em pdf ou docx. Existe uma condição, já que de cada extensão o texto é removido de maneira diferente, com módulos *Python* diferentes.

```
def object_create(diretorio):
arquivos = import_arquivos(diretorio)
textos = []
nomes = []
erros = []
for arq in arquivos:
    docx = []
    if (arq.endswith(".docx")):
        aux = Document(arq)
        nomes.append(arq)
        for doc in aux.paragraphs:
            temp = doc.text
            temp =re.sub('\s+', ' ', unicodedata.normalize("NFKD",temp)
                        .encode('ASCII','ignore')
                        .decode('ASCII')).lower().strip()
            docx.append(temp)
        textos.append(' '.join(docx))
    else:
        try:
            temp= parser.from_file(arq)['content']
            temp = temp.lower()
            temp = temp.replace("\n"," ")
            nomes.append(arq)
            textos.append(temp)
        except Exception as e:
            try:
                temp = open(arq, 'rb')#insere no python
                pdf = PyPDF2.PdfFileReader(temp)#cria objeto
                for j in range (pdf.getNumPages()):
                    temp = pdf.getPage(j).extractText()#retira o texto
                    temp = temp.lower()
                    textos = textos.replace("\n"," ")
                    textos = textos.replace("   "," ")
                    nomes.append(arq)
                    textos.append(temp)
```

```
                except Exception as e:
                      erros.append([e,arq])
                continue
    return textos,nomes,erros
```

Na extração de docx, o modulo Document extrai parágrafos a cada iteração, por essa razão foi feito um join após cada extração de parágrafos.
Nas duas extrações já começa uma limpeza inicial dos dados reduzindo a caixa das letras, espaços em branco e linhas extras.
O retorno dessa função consiste na lista do corpus dos textos a serem comparados, na lista dos títulos desses corpus e nos erros que possam ocorrer na extração por arquivo corrompido ou outros.
*A critério de ajustes de erros, foram criadas exceções e retornadas a fim de ajustes do softwares caso necessário, entretanto não é retornado para o usuário.

2.1.3 Limpeza

A função **limpeza**, como o nome deixa explícito, limpa do texto as stopwords, que são palavras que se repetem muito, como preposições e verbos auxiliares, podendo interferir na similaridade. O retorno dessa função é a lista de textos limpos e padronizados e os títulos dos documentos.

```
def limpeza(diretorio):
    textos, titulos, erros = object_create(diretorio)
    stop_words = stopwords.words('portuguese') + list(punctuation)
    for t in range(len(textos)):
        textos[t] = ' '.join([word for word in textos[t].split(" ")
                     if word not in stop_words])
    return textos, titulos #,erros
```

2.1.4 Similaridade

A função **similaridade** cria um modelo de acordo com o conjunto de dados chamado corpus que pode ser pequeno, médio ou grande, definido como **"pt_core_news_*i*"** onde i pode ser sm (small), md (medium) ou lg (large).

```
    def similaridade(diretorio):
    textos, titulos = limpeza(diretorio)
    similaridade = pd.DataFrame(columns = ['Título_1', "Similaridade",'Título_2'])
    Título_1 = []
    Título_2 = []
    Similaridade = []
    nlp = spacy.load ("pt_core_news_md")
```

```
    for i in range(len(textos)):
        for j in range(len(textos)):
            Título_1.append(titulos[i])
            Título_2.append(titulos[j])
            Similaridade.append(nlp(textos[i]).similarity(nlp(textos[j])))
    similaridade['Título_1'] = Título_1
    similaridade['Título_2'] = Título_2
    similaridade["Similaridade"] = Similaridade
    similaridade.to_excel(diretorio +'/textos_similaridade.xlsx')
```

Para cada texto é feita uma comparação e o retorno função é um dataframe com 3 colunas que são os títulos comparados e a similaridade entre eles.

3 Documentações externas

https://www.nltk.org/
https://python-docx.readthedocs.io/en/latest/#
https://pypi.org/project/PyPDF2/
https://github.com/chrismattmann/tika-python
https://spacy.io/models/pt