
Especialização em Inteligência Artificial (EiA - IFMG-OP) - Recuperação de Informação

Atividade 3 - Sistema de Recuperação de Informação sobre Enfermagem

Prof. Moisés - Aluno: Fernando dos Santos Alves Fernandes

Contexto

Inicialmente, o contexto definido para o Sistema de Recuperação de Informação implementado foi a **Estomaterapia**, que é uma especialidade de enfermagem voltada para a assistência de pacientes com estomias, fístulas, tubos, catéteres, drenos, feridas agudas e crónicas (diabetes, úlceras, hérnias) e incontinências urinária e anal. O objetivo foi construir um sistema de recuperação de informação capaz de permitir a busca de informações sobre o tema, como conceitos, definições, tipos de procedimentos cirúrgicos associados, tratamentos, links de instituições que oferecem essa especialização, entre outras informações. No entanto, para tornar o sistema mais abragente e possibilitar melhor o processo de avaliação, o contexto foi ampliado para **Enfermagem**.

Coletor

A estratégia de busca para aprofundamento das páginas encontradas foi a ***Breadth-First Search*** (Busca em Largura), com limite de profundidade (**max_depth**). Nesse coletor, simples, as tags de texto utilizadas foram apenas a <h1> e <h2>. Quanto às requisições, redirecionamentos de páginas foram permitidos, por meio do parâmetro **allow_redirects=True**. O parâmetro **headers** também foi utilizado, para evitar que o coletor fosse bloqueado por determinadas páginas, como as do Governo. Os resultados de algumas coletas de teste (apenas as URLs das páginas coletadas) podem ser vistos em arquivos '.txt', que acompanham os arquivos fonte do coletor e do programa principal no link da Atividade 1. A seguir, o código de implementação do coletor.

```
import requests
from bs4 import BeautifulSoup
from url import Url
```

```python
class Coletor:
    #def __init__(self, codigo) -> None:
    def __init__(self, codigo, max_depth=2) -> None:
        self.codigo = codigo
        self.max_depth = max_depth # Máximo nível de aprofundamento na busca por novas páginas.
        self.urls = {}
        self.objects_url = []
        self.headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/91.0.4472.124 Safari/537.36 Edg/91.0.864.59"} # Para não ser bloqueado.

    def add_url(self, url) -> None:
        if url not in self.urls.keys(): # Não permite incluir urls repetidas!
            self.urls[url] = False # Chegando agora! Ainda não foi processada!
            print(f'[INFO]: add_url                  - new url added = {url}.')

    def extract_information(self) -> None:
        good_urls = [url for url in self.urls.keys() if not self.urls[url]]
        for url in good_urls:
            response = requests.get(url, allow_redirects=True) # Fazendo requisição na página
            if response.status_code == 200:
                self.objects_url += [Url(url, BeautifulSoup(response.text, 'html.parser'))]
                self.urls[url] = True
            else:
                print(f'[coletor  ] Falha ao carregar a página: {response.status_code}')

    def bfs_extract_information(self, depth) -> None: # Breadth-First Search (Busca em Largura)
        print(f'[INFO]: bfs_extract_information - depth = {depth}.')
        print(f'[INFO]: bfs_extract_information - urls size = {len(self.urls)}.')
        if depth > self.max_depth:
            return

        good_urls = [url for url in self.urls.keys() if not self.urls[url]] # Seleciona os itens do dicionário,
cujos valores são False (não visitadas: {'url2': False, 'url3': False, 'url4': False, 'url5': False}).
        print(f'[INFO]: bfs_extract_information - good_urls size = {len(good_urls)}.')
        for url in good_urls: # good_urls = ['url2', 'url3', 'url4', 'url5']
            print(f'[INFO]: bfs_extract_information - current url in good_urls = {url}.')
            try:
                response = requests.get(url, headers = self.headers, allow_redirects=True)#, verify=False) #
```

```
Fazendo requisição na página
                            # Verifica se houve redirecionamento
                if response.history:
                    print(f'[INFO]: bfs_extract_information - redirect detected ({url}).')
                if response.status_code == 200:
                    print(f'[INFO]: bfs_extract_information - response.status_code ({url}) =
{response.status_code}.')
                    url_obj = Url(url, BeautifulSoup(response.text, 'html.parser'))
                    self.objects_url += [url_obj] # Cada item dessa lista tem a url e o conteúdo das tags <title>
(título da página), <h2> (conteúdos de texto), <a> (links encontrados).
                    self.urls[url] = True # Valor da url no dicionário é alterado para True (página visitada!)
                    new_urls = [link['href'] for link in url_obj.links if ('href' in link.attrs) and
(link['href'].startswith('http')) and (link['href'] not in self.urls.keys())]
                    print(f'[INFO]: bfs_extract_information - new_urls size = {len(new_urls)}.')
                    for url in new_urls:
                        print(f'[INFO]: bfs_extract_information - url = {url}.')
                        self.add_url(url)
                    print(f'[INFO]: bfs_extract_information - next recursive call, depth = {depth + 1}.')
                else:
                    print(f'[ERROR]: bfs_extract_information - response error = {response.status_code}.')
            except Exception as e:
                print(f'[ERROR]: bfs_extract_information - response error = [{url}] -> {e}.')
        self.bfs_extract_information(depth + 1)

    def print_results(self): # Somente as URLs encontradas!
        print(f'print_results:\n')
        url_set = set(self.objects_url)
        for obj in url_set:
            print(f'\turl = {obj.url}')

    def save_results(self): # Somente as URLs encontradas!
        filename = self.codigo
        with open(f"index-{filename}.txt", 'w', encoding='utf-8') as file:
            url_set = set(self.objects_url)
            for obj in url_set:
                file.write(obj.url+'\n')
```

Indexador

No mesmo contexto definido na implementação do coletor (Atividade 1), foi implementado o indexador, utilizando como abordagem uma lista invertida. Para a avaliar a força da palavra ou termo, foram considerados os valores de f(K), F(K) e n(K), implementados em sala de aula, em que o K é a chave ou o termo indexado na lista invertida. Para esse trabalho, a métrica TF-IDF também foi calculada para cada um dos tokens. O peso f(K), também conhecido como TF (*Term Frequency*, TF(t,d)) é o número de ocorrências do termo t no documento d. F(K) é o total de ocorrências do termo t (ou chave K), considerando todos os documentos em que ele é encontrado. O peso n(K), que também pode ser encontrado na literatura como DF(t), ou *Document Frequency* do termo t, corresponde ao número de documentos em que a chave K ocorre. O IDF(t) (*Inverse Document Frequency*, do termo t) é o peso do termo que considera o número de documentos coletados (N) e o DF do termo e pode ser calculado como ***TF-IDF*** $= f(k) * idf(k) = f(k) * log[N/n(k)]$.

A implementação completa do indexador pode ser vista a seguir e encontra-se disponível em Atividade 2. A versão mais atuailzada do indexador, apresentada a seguir, encontra-se disponível em Atividade 3.

```
import json
import math
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt')

class Indexador:
    def __init__(self, coletor) -> None:
        self.coletor = coletor
        #self.tokenized_titles = []
        self.stop_words = set(stopwords.words('portuguese'))
        self.inverted_index = {}
        self.F = {}

    def inverted_index_generator(self):
        for object in self.coletor.objects_url:
            tokenized_titles = []
            for title in object.titles: # Os h2!
                tokens = [token for token in [token.lower().replace('\u200b', '') for token in
nltk.word_tokenize(title.text) if token.lower() not in self.stop_words] if token.isalnum()] # Lista de palavras
```

```
relevantes!
                tokenized_titles.extend(tokens)

            f = {}
            for token in tokenized_titles:
                if token not in f:
                    f[token] = 1
                    if token not in self.F:
                        self.F[token] = 0
                else:
                    f[token] += 1

            # Term Frequency:
            #         / 1 + log(f) if fi,j(k) > 0,
            #   tf = {
            #         \ 0, otherwise
            #
            # Inverse Document Frequency:
            #   idf = log N / ni
            #   N = len((self.coletor.objects_url)

            for token in tokenized_titles: # f(k), F(k), n(k), idf
                self.inverted_index.setdefault(token, {}).update({object.url: [f[token], self.F[token], 0, 0]}) #
Link e peso do token!

            for token in self.inverted_index.keys():
                if token in f:
                    self.F[token] += f[token]

    def update_F(self):
        for token in self.inverted_index.keys():
            for object_url in self.inverted_index[token].keys():
                self.inverted_index[token][object_url][1] = self.F[token] # F(k)
                self.inverted_index[token][object_url][2] = len(self.inverted_index[token].keys()) # n(k)
                self.inverted_index[token][object_url][3] = self.inverted_index[token][object_url][0] *
math.log(len(self.coletor.objects_url)/self.inverted_index[token][object_url][2]) # TF-IDF = f(k) * idf(k) = f(k)
* log [N / n(k)].
```

```
    def save_index(self):
        filename = self.coletor.codigo
        with open(f"index-{filename}.json", 'w') as file:
            json.dump(self.inverted_index, file, indent=4)
        print(f'[indexador] index-{filename}.json = {len(self.inverted_index.keys())}')
```

Buscador

A seguir, a implementação do buscador. Para permitir o ranqueamento dos resultados, considerando os diferentes termos de uma busca, foi utilizada a métrica **TF-IDF** implementada no indexador. Para cada termo da busca, os itens do índice invertido relacionados ao termo são ordenados decrescentemente e adicionados à lista de *links* relevantes. Duas estratégias de processamentos de consultas foram implementados, a **Interseção**, que considera apenas os *links* relevantes que aparecem simultaneamente nos resultados de todos os termos da busca; e a **União**, que consideram todos os *links* relevantes de todos os termos da busca. Na classe do buscador também são implementados os métodos para o cálculo da Precisão (***precision***) e da Revocação (***recall***), que são utilizados para avaliar a qualidade dos resultados do sistema de recuperação de informação.

```
import json
import nltk

class Buscador: # 1 para N
    def __init__(self, index_file) -> None:
        with open(index_file, 'r') as file:
            self.inverted_index = json.load(file)

    def search(self, query) -> set:
        query_tokens = nltk.word_tokenize(query)
        relevant_links = set()
        # É necessário remover as stop words das queries? Aparentemente, não precisa!
        for token in query_tokens:
            if token in self.inverted_index:
                relevant_links.update(self.inverted_index[token])
        return relevant_links

    def new_search(self, query, query_type=1) -> set:
        query_tokens = nltk.word_tokenize(query)
```

```
        token_relevant_links = {}
        # print(f'{self.inverted_index}')
        for token in query_tokens:
            #print(f'self.inverted_index[token] = {self.inverted_index[token]}')
            # Preciso ranquear self.inverted_index[token]!!!
            if token in self.inverted_index:
                #ordered_inverted_index = dict(sorted(self.inverted_index[token].items(), key=lambda item:
item[1]))
                #print(f'\nOrdered inverted_index[{token}]["tf-idf"] =
{dict(sorted(self.inverted_index[token].items(), key=lambda item: item[1][3], reverse=True))}')
                token_relevant_links[token] = set()
                #token_relevant_links[token].update(self.inverted_index[token])
                # O ranqueamento está sendo feito aqui, pelo 'tf-idf'.
                token_relevant_links[token].update(dict(sorted(self.inverted_index[token].items(), key=lambda
item: item[1][3], reverse=True)))
                #print(f'token_relevant_links[token] = {token_relevant_links[token]}')
        #print(f'\n{list(token_relevant_links.values())}')
        list_relevant_links = list(token_relevant_links.values())
        #print(f'{(list_relevant_links)}')
        final_relevant_links = []
        if len(list_relevant_links) > 0:
            final_relevant_links = list_relevant_links[0]
            for set_relevant_links in list_relevant_links[1:]:
                match query_type:
                    case 1:
                        final_relevant_links = final_relevant_links.intersection(set_relevant_links)
                    case 2:
                        final_relevant_links = final_relevant_links.union(set_relevant_links)
        return final_relevant_links

    def precision(self, relevants, results):
        results_size = len(results)
        intersection_size = len([link for link in results if link in relevants])
        return intersection_size / results_size

    def recall(self, relevants, results):
        relevants_size = len(relevants)
```

```
        intersection_size = len([link for link in results if link in relevants])
        return intersection_size / relevants_size
```

O Sistema de RI

A seguir, o código principal, com a lista de URLs iniciais relacionadas ao contexto da Enfermagem. É aqui também que são instanciados os objetos das classes **Coletor**, **Indexador** e **Buscador** apresentadas acima. O usuário interage com o *prompty de comando*, informando os termos da pesquisa que deseja realizar. Utilizando-se do arquivo indexado, o buscador recupera as URLs relevantes, de acordo com os termos da consulta. Se a variável **query_type** for igual a **1**, a estratégia de processamento da busca será uma **Interseção**. Caso contrário (**query_type = 2**), os resultados serão a **União** dos conjuntos de *links* relevantes de todos os termos da pesquisa. A variável **assessment** define se o sistema deve ser avaliado, considerando a lista de resultados da busca e a lista de *links* considerados relevantes para aquela determinada busca. Caso o seu valor ser ***True***, os valores das métricas ***Precision*** e ***Recall*** são apresentados.

```
from coletor import Coletor
from indexador import *
from buscador import Buscador

# urls = ['https://sobest.com.br/', 'https://pt.wikipedia.org/wiki/Estomaterapia',
'https://www.estomaterapiauerj.com.br/', 'https://www.souenfermagem.com.br/fundamentos/estomia-e-ostomia/',
'https://www.cuf.pt/mais-saude/ostomia-o-que-e-e-em-que-casos-e-necessaria', 'https://www.hnipo.org.br/conheca-
tudo-sobre-a-estomaterapia-do-hospital-nipo-brasileiro-e-seus-beneficios-aos-
pacientes/#:~:text=O%20que%20%C3%A9%20estomaterapia%3F,e%20incontin%C3%AAncias%20urin%C3%A1ria%20e%20anal',
'https://nutritotal.com.br/pro/material/ostomia-o-que-e-e-quais-sao-os-cuidados/']

urls = ['https://www.paho.org/pt/topicos/enfermagem', 'https://pt.wikipedia.org/wiki/Enfermagem',
'https://www.corenmg.gov.br/', 'https://www.cofen.gov.br/enfermagem-em-numeros/', 'https://enfermfoco.org/',
'https://www.gov.br/capes/pt-br/acesso-a-informacao/acoes-e-programas/avaliacao/sobre-a-avaliacao/areas-
avaliacao/sobre-as-areas-de-avaliacao/colegio-de-ciencias-da-vida/ciencias-da-saude/enfermagem',
'https://cmmg.edu.br/enfermagem/']

contexto = 'enfermagem'
coletor = Coletor(contexto, 2) # Opcionalmente, pode-se passar o nível máximo de aprofundamento, max_depth.
for url in urls:
    coletor.add_url(url)
```

```
coletor.bfs_extract_information(0)
coletor.print_results()
coletor.save_results()

indexer = Indexador(coletor)
indexer.inverted_index_generator()
indexer.update_F()
indexer.save_index()

query = input("[buscador ] O que você deseja pesquisar? ")
searcher = Buscador("index-"+contexto+".json")
query_type = 1 # 1 = Interseção; 2 = União.
assessment = False

relevantes_curso_enfermagem = ["https://www.educamaisbrasil.com.br/cursos-e-faculdades/enfermagem",
            "https://querobolsa.com.br/cursos-e-faculdades/enfermagem",
            "https://querobolsa.com.br/cursos-e-faculdades/enfermagem",
            "https://cmmg.edu.br/enfermagem/"
]
relevants = relevantes_curso_enfermagem

results = searcher.new_search(query, query_type)
print(f'\n[buscador ] Resultados: ')
for r in results:
   print(f"\t* {r}")

if assessment and len(relevants) > 0 and len(results) > 0:
    print(f'\n[buscador ] Avaliação do Sistema de Recuperação de Informação:')
    print(f'[buscador ]\tPrecisão = {searcher.precision(relevantes_curso_enfermagem, results)}')
    print(f'[buscador ]\tRevocação = {searcher.recall(relevantes_curso_enfermagem, results)}')
```

O Sistema de RI com uma 'cara' mais bonitinha

Utilizando o *framework* Streamlit, foi implementada uma versão *web* do Sistema de Recuperação de Informação sobre Enfermagem. As funcionalidades são as mesmas do sistema já apresentado, no entanto, nesse caso, não são realizadas as etapas de coleta e indexação, uma vez que o objetivo foi implementar uma interface de consulta mais amigável para o usuário. A ferramenta ainda permite que o usuário

selecione o tipo de consulta (opção associada à estratégia de processamento dos resultados): **todas as palavras** (interseção) ou **qualquer das palavras** (união); e defina se quer uma avaliação do sistema (*checkbox* com rótulo "**Avaliar o Sistema de RI**"). Caso o usuário queira avaliar o sistema, é possível definir a lista de *links* que considera relevantes e deveriam "aparecer" nos resultados. Com base nessa lista informada pelo usuário, as métricas de precisão e revocação são calculadas e apresentadas. O código completo é apresentado a seguir e encontra-se disponível em Atividade 3.

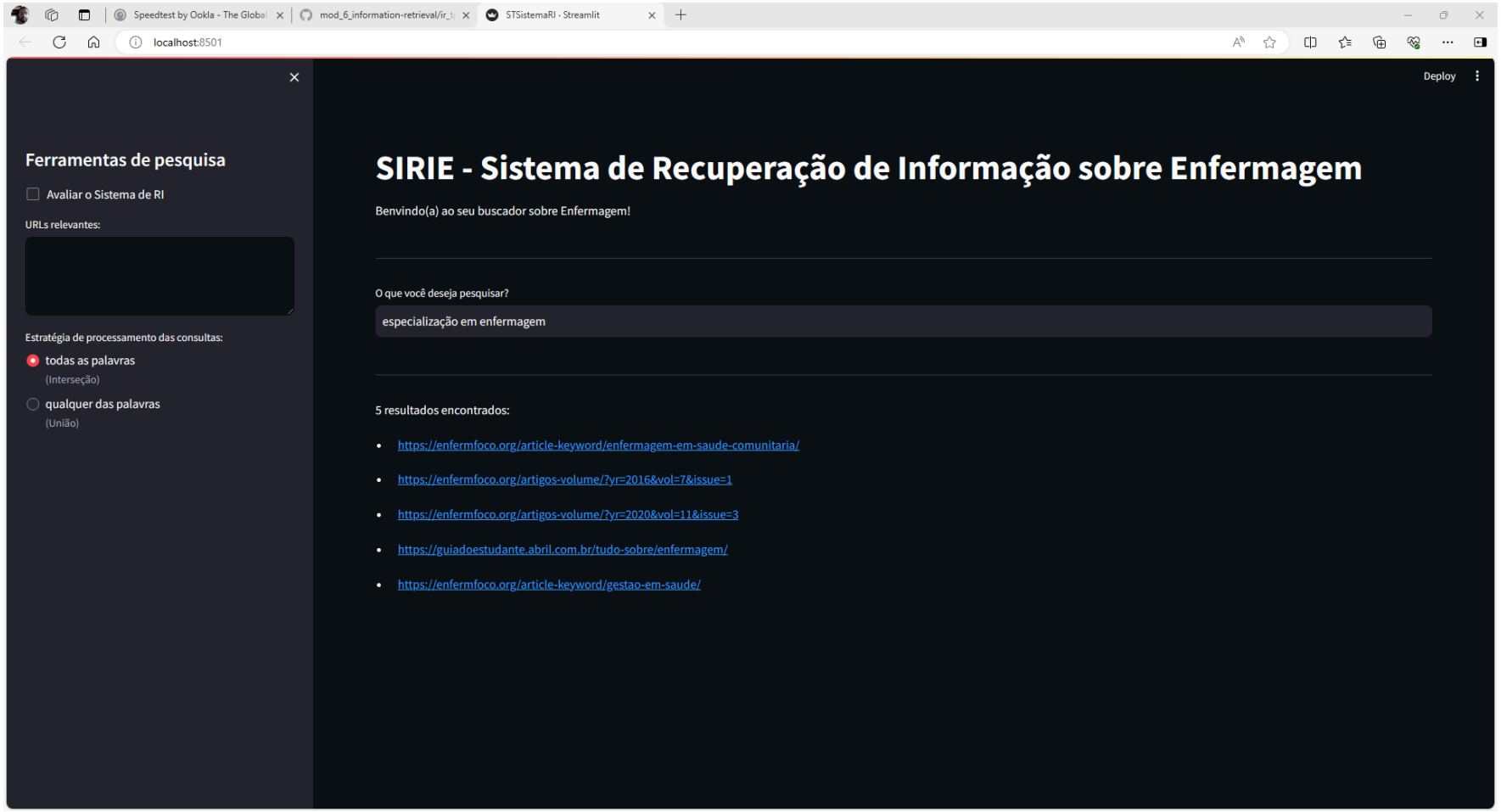
```
from coletor import Coletor
from indexador import *
from buscador import Buscador
import streamlit as st
import re

def to_assessment():
    print(f'{st.session_state.checker}')

def type_of_query():
    print(f'{str(st.session_state.radio)}')

st.title('SIRIE - Sistema de Recuperação de Informação sobre Enfermagem')
st.write('Benvindo(a) ao seu buscador sobre Enfermagem!')
st.markdown('--------')

contexto = 'enfermagem'
query = st.text_input("O que você deseja pesquisar? ")

with st.sidebar:
    st.sidebar.title("Ferramentas de pessquisa")
    assessment = st.sidebar.checkbox("Avaliar o Sistema de RI", value=False, on_change=to_assessment,
key="checker")
    relevant_urls = st.sidebar.text_area(label="URLs relevantes:")

    query_type_option = st.sidebar.radio("Estratégia de processamento das consultas:",
                                         options=["todas as palavras", "qualquer das palavras"],
                                         captions=["(Interseção)", "(União)"],
                                         on_change=type_of_query,
                                         key="radio")
    if (query_type_option == "todas as palavras"): #query_type = 1 # 1 = Interseção; 2 = União.
```

```
        query_type = 1
    else:
        query_type = 2

st.markdown('--------')

#relevants = list(relevant_urls.split(sep=','))
relevants = list(re.split(',| |',relevant_urls)) # usando regex!

if query or st.button('Buscar'):
    searcher = Buscador("index-"+contexto+".json")
    results = searcher.new_search(query.lower(), query_type) # passar o tipo de query!
    st.write(f"Resultados: ")
    for r in results:
       st.markdown('- ' + r)
    if assessment and len(relevants) > 0 and len(results) > 0:
        st.write("Avaliação do Sistema de Recuperação de Informação:")
        st.markdown('- Precisão = ' + str(searcher.precision(relevants, results)))
        st.markdown('- Revocação = ' + str(searcher.recall(relevants, results)))
```

Resultados de algumas consultas

A seguir, os resultados encontrados para a pesquisa "**tratamento de feridas**".

A busca, agora, foi por "**especialização em enfermagem**". Observa-se que, utilzando as "*Ferramentas de pesquisa*", é possível, em "*Estratégia de processamento das consultas*", selecionar se os resultados devem ser URLs contendo, simultaneamente, todos os termos da pesquisa (excluindo-se as *stop_words*) ou se devem retornar todos os *links* em que qualquer um dos termos aparece. As figuras a seguir exibem os resultados em ambos os casos. Para a opção a *Interseção*, o sistema encontrou 5 resultados, enquanto para a *União*, o sistema recuperou 1212 *links* relacionados ao termos da consulta.


Speedtest by Ookla - The Global
mod_6_information-retrieval/ir_t
STSistemaRI · Streamlit
localhost:8501
Deploy
Ferramentas de pesquisa
Avaliar o Sistema de RI
URLs relevantes:
Estratégia de processamento das consultas:
todas as palavras
(Interseção)
qualquer das palavras
(União)
SIRIE - Sistema de Recuperação de Informação sobre Enfermagem
Benvindo(a) ao seu buscador sobre Enfermagem!
O que você deseja pesquisar?
especialização em enfermagem
5 resultados encontrados:
https://enfermfoco.org/article-keyword/enfermagem-em-saude-comunitaria/
https://enfermfoco.org/artigos-volume/?yr=2016&vol=7&issue=1
https://enfermfoco.org/artigos-volume/?yr=2020&vol=11&issue=3
https://guiadoestudante.abril.com.br/tudo-sobre/enfermagem/
https://enfermfoco.org/article-keyword/gestao-em-saude/

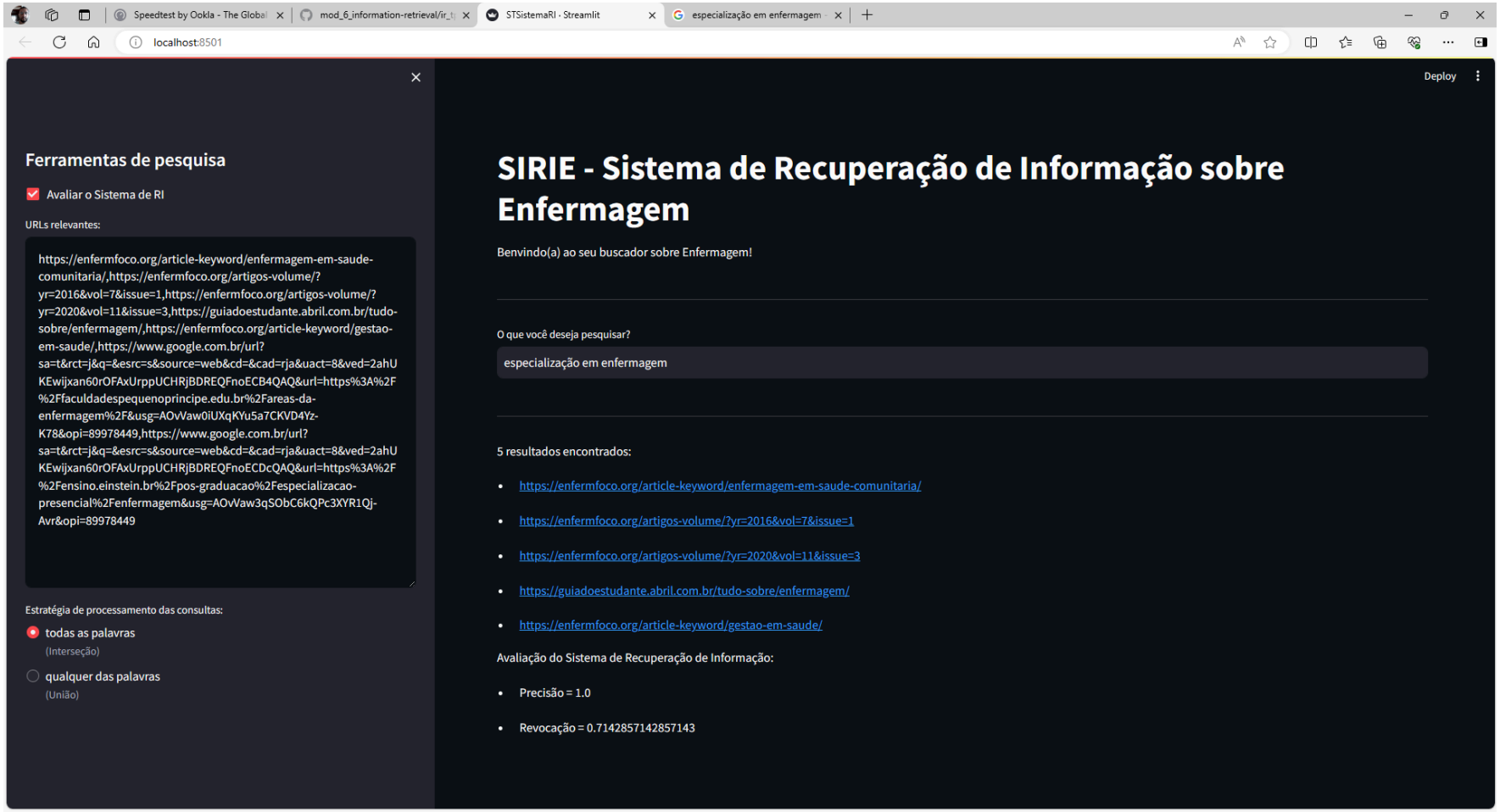
Outro recurso disponível permite a "Avaliação do Sistema de RI", uma vez que o *checkbox* correspondente esteja selecionado. Com base na lista de *URLs relevantes* para uma dada consulta, o sistema calcula as métricas *Precisão* e *Revocação*.

À medida em que vamos atualizando a lista de *URLs relevantes*, as métricas de *Precisão* e de *Revocação* vão sendo atualizadas.

A *Revocação* igual a 1.0 indica que todas as *URLs relevantes* foram recuperadas e encontram-se na lista de resultados encontrados. A *Precisão* menor do que 1.0 indica que os resultados apresentam **Falsos positivos**, ou seja, o sistema recuperou documentos que não deveriam ter sido encontrados.

Quando a lista de *resultados encontrados* pelo sistema coincide com a lista de *URLs relevantes*, os valores de *Precisão* e de *Revocação* são iguais a 1.0, indicando que o sistema tem o melhor desempenho possível. Todas as *URLs relevantes* foram encontradas e somente elas foram recuperadas.

Quando os *resultados encontrados* não incluem todas as *URLs relevantes*, a *Precisão* é igual a 1.0, porque todos os *links* recuperados se encontram na lista de *URLs relevantes*. No entanto, a *Revocação* é um valor menor do que 1.0.


Speedtest by Ookla - The Global
mod_6_information-retrieval/ir_t
STSistemaRI - Streamlit
especialização em enfermagem
localhost:8501
Deploy
Ferramentas de pesquisa
Avaliar o Sistema de RI
URLs relevantes:
https://enfermfoco.org/article-keyword/enfermagem-em-saude-comunitaria/,https://enfermfoco.org/artigos-volume/?yr=2016&vol=7&issue=1,https://enfermfoco.org/artigos-volume/?yr=2020&vol=11&issue=3,https://guiadoestudante.abril.com.br/tudo-sobre/enfermagem/,https://enfermfoco.org/article-keyword/gestao-em-saude/,https://www.google.com.br/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwijxan60rOFAxUrppUCHRjBDREQFnoECB4QAQ&url=https%3A%2F%2Ffaculdadespequenoprincipe.edu.br%2Fareas-da-enfermagem%2F&usg=AOvVaw0iUXqKYu5a7CKVD4Yz-K78&opi=89978449,https://www.google.com.br/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwijxan60rOFAxUrppUCHRjBDREQFnoECDcQAQ&url=https%3A%2F%2Fensino.einstein.br%2Fpos-graduacao%2Fespecializacao-presencial%2Fenfermagem&usg=AOvVaw3qSObC6kQPc3XYR1Qj-Avr&opi=89978449
Estratégia de processamento das consultas:
todas as palavras
(Interseção)
qualquer das palavras
(União)
SIRIE - Sistema de Recuperação de Informação sobre Enfermagem
Benvindo(a) ao seu buscador sobre Enfermagem!
O que você deseja pesquisar?
especialização em enfermagem
5 resultados encontrados:
https://enfermfoco.org/article-keyword/enfermagem-em-saude-comunitaria/
https://enfermfoco.org/artigos-volume/?yr=2016&vol=7&issue=1
https://enfermfoco.org/artigos-volume/?yr=2020&vol=11&issue=3
https://guiadoestudante.abril.com.br/tudo-sobre/enfermagem/
https://enfermfoco.org/article-keyword/gestao-em-saude/
Avaliação do Sistema de Recuperação de Informação:
Precisão = 1.0
Revocação = 0.7142857142857143