

INFORMATION RETRIEVAL AND WEB SEARCH

ASSIGNMENT-8

SUBMITTED BY:

NAME: Christine Odero

STUDENT ID: U00676102.

Email: codero@memphis.edu

DUE DATE: 11/29/2021

SUBMITTED TO: DR VASILE RUS

Table of Contents

[Table of Contents 2](#_Toc89689361)

[Introduction 3](#_Toc89689362)

[Approach 3](#_Toc89689363)

[Design 3](#_Toc89689364)

[Implementation 3](#_Toc89689365)

[RESULTS 11](#_Toc89689366)

[Future work 18](#_Toc89689367)

[Reference 19](#_Toc89689368)

# Introduction

Information retrieval is the science of searching for information in a document. Which involves searching for documents themselves, searching for the metadata that describes the dataset and for a database of texts, images, or sounds. There are many ways of retrieving data including reading into books, watching videos, web searching or listening to audio tapes.

Web Searching has grown with the incremental increase of computers and computer related devices such as smart watches, tablets, and phones. With this in mind, we use search engines regularly. When we have queries, we can use a search engine like Google to retrieve the most relevant answer.

To achieve this site like google index information for easy retrieval for their user. The process of indexing involves crawling into various websites, extracting essential information from the document, going ahead, and indexing them according to the token.

Most of the queries are text-based. In this case the University of Memphis has a search engine enhanced by google. As a Data Science student, we saw the need to create our own custom search that is not dependent on google.

# Approach

1. Extracting the 10,000 text files from Memphis.edu.
2. Preprocess the 10,000 txt files.
3. Indexing the files.
4. Create a search.

### Design

# Implementation

#### 1.1 Data preparation getlinks.py

1.1.2 Dependencies:

import urllib.request, urllib.parse, urllib.error

from bs4 import BeautifulSoup

from christineAssign6 import fetch

import logging

1. [UrlLibray](https://docs.python.org/3/library/urllib.html#module-urllib) This is an inbuilt library that is used for handling url/link requests. Here the imported methods are:

* [urllib.request](https://docs.python.org/3/library/urllib.request.html#module-urllib.request) for opening and reading URLs
* [urllib.error](https://docs.python.org/3/library/urllib.error.html#module-urllib.error) containing the exceptions raised by [urllib.request](https://docs.python.org/3/library/urllib.request.html#module-urllib.request)
* [urllib.parse](https://docs.python.org/3/library/urllib.parse.html#module-urllib.parse) for parsing URLs. (urllib — URL handling modules, 2021)

1. [Beautiful-soup](https://beautiful-soup-4.readthedocs.io/en/latest/) This is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree. (beautiful-soup-4, 2021)
2. [Logging module](https://docs.python.org/3/library/logging.html?highlight=logging#module-logging) this an inbuilt python module used to log out what the program is executing
3. Fetch function - this is a function that takes in URL links, extracts the text information from html,pdf,or docs and saves this information as a “.txt” file (logging — Logging facility for Python, 2021).

1.1.2 Code explanation

The links home function takes in a link ([https://memphis.edu](https://memphis.edu/)) Which is a string and returns a list of all individual links using anchor tag a from the main Memphis site.

def links\_home(link: str):

html = urllib.request.urlopen(link).read()

return (BeautifulSoup(html, "html.parser"))('a')

The get\_href gets all tags from links\_home function and returns a list of links that are saved inside 'href' attribute

def get\_href(tags: list):

return [tag.get('href') for tag in tags]

The more links function takes in link and generates additional links from the list of links generated by get\_href.

The cleaning links method removes ID links from all the links generated

def morelinks(link: str):

return get\_href(links\_home(link))

def cleaninglinks(links: list):

return [

link

for link in links

if not ((link.startswith("#")) or (link.startswith('//')))

]

1.1.3 Driver code

if \_\_name\_\_ == "\_\_main\_\_":

total = get\_href(links\_home("https://www.memphis.edu/"))

total=cleaninglinks(total)

print(total)

for link in total:

logging.info(f"Getting data from {link}")

try:

fetch(link)

except:

logging.info(f"Error Getting data from {link}")

continue

1. Inside the total variable, a function call is made get\_href to get all links with ‘href’ attribute from a list of links. This is achieved by calling links home which returns a list of all links with “a” anchor tag.
2. The above links are stored in a variable named total, by calling cleaning links(total) cleans all the links by removing all the ID related links.
3. The resulting cleaned links are then passed into fetch () function which generates “.txt” files

#### 1.2 Preprocessing the data

1.2.1 Dependencies:

import re

import string

from spacy.lang.en.stop\_words import STOP\_WORDS as stop\_words

from nltk.tokenize import word\_tokenize

from nltk.stem import PorterStemmer

1. [Spacy](https://spacy.io/) - A open-source software library for advanced Natural Language Processing, used this library because it has all stop word and is also faster (ines,2021)
2. [Ntlk](https://www.nltk.org/) - Library is a suite that contains libraries and programs for statistical language processing. (Natural Language Toolki, 2021)
3. [Re](https://docs.python.org/3/library/re.html) - regular expression is a special sequence of characters that helps to match or find the strings (Regular expression operations, 2021)

1.2.2 Code explanation

1. Converting to lowercase

def lower(text: str):

return [text.lower()]

1. Removing Numbers

def remove\_numbers(text: str):

return re.sub(r"\d+", "", text)

1. Removing punctuations

def remove\_punctuations(text):

return text.translate(str.maketrans("", "", string.punctuation))

1. Removing whitespaces

def removing\_whitespaces(text: str):

return text.strip()

1. Removing stop\_words

def removing\_stop\_words(text: str):

token = word\_tokenize(text)

return [i for i in token if i not in stop\_words]

1. Stemming

def stemming(text):

return [stemmer.stem(txt) for txt in text]

1. Preprocessing “.txt” files

def read\_text\_file(file\_path):

with open(file\_path, "r") as f:

txt = \_extracted\_from\_read\_text\_file\_3(f)

with open(file\_path, "w") as f:

f.write(txt)

return txt

def \_extracted\_from\_read\_text\_file\_3(f):

result = f.read()

# to lower case

result = lower(result)

result = remove\_numbers(result)

result = remove\_punctuations(result)

result = removing\_whitespaces(result)

result = removing\_stop\_words(result)

result = stemming(result)

return result

#### 

#### 1.3 Loading preprocessed data into Memory for indexing

##### 1.3.2 Creating memory Holder

###### 1.3.2.1 Dependencies:

from collections import Counter

from dataclasses import dataclass

from preprocessor import cleaning\_query

1. Counter - The Counter class in Python3 is a specific form of object dataset that comes with the collection’s module. The Collections module provides users with specialized container data types as an alternative to Python's built-ins such as dictionaries, lists, and tuples.
2. Data class -as a utility tool to make structured classes specially for storing data. These classes hold certain properties and functions to deal specifically with the data and its representation.
3. cleaning Query - this is a preprocessor function imported from preprocessor.py

Code Explanation

class Data:

id:int

text : str

url: str

Class Data has three properties associated to each “.txt” document namely:

Id: this is meant to associate each document with a special identifier

Text: this is text that is contained inside each “.txt” file

URL: this refers to the original source of the file.

Methods

def analyze(self):

self.term\_frequencies = Counter(cleaning\_query(self.text))

1. analyze-This is meant to count the frequency of individual token/term

def term\_frequency(self, term):

return self.term\_frequencies.get(term, 0)

1. Term frequency-Acts as helper function to analyze method
2. \_\_repr\_\_ this returns URL for individual document, since we are only interested

In getting url where certain words are highly common

def \_\_repr\_\_(self) -> str:

return f'{self.url}'

##### 1.4 Loading data from “.txt” files to memory

Loading.py

def load\_documents():

path = "./christine/results"

output = r"results/"

os.chdir(output)

# iterate through all file

number = 1

for file in os.listdir():

# Check whether file is in text format or not

if file.endswith(".txt"):

file\_path = f"{path}/{file}"

with open(file\_path, 'r') as f:

p = f.read()

link=p.split()[0]

text = (p.split()[1:])

txt = " ".join(text)

number= number + 1

yield Data(id=number, text=txt, url = link)

Load\_document function loads “.txt” files, opens individual file loads txt,then saves into Data class using yield Generator.

##### 1.5 Indexing the loaded data, Searching, and ranking

###### Indexing

def \_\_init\_\_(self):

self.index = {}

self.documents = {}

def index\_document(self, document):

if document.id not in self.documents:

self.documents[document.id] = document

# frequency counts when we index our data

document.analyze()

for token in cleaning\_query(document.text):

if token not in self.index:

self.index[token] = set()

self.index[token].add(document.id)

The method index document indexes each document using the document Id and goes ahead to also index all the tokens into an index hand in hand with the documents. Such that each token has a list of documents where it has appeared.

###### searching

Since all tokens are indexed, searching becomes a matter of analyzing the query text with the same analyzer as we applied to the documents; this way we will end up with tokens that should match the tokens we have in the index. For each token, we will do a lookup in the dictionary, finding the document IDs that the token occurs in. We do this for every token, and then find the IDs of documents in all these sets (i.e. for a document to match the query, it needs to contain all the tokens in the query). We will then take the resulting list of documents.

###### ranking

def rank(self, analyzed\_query, documents):

results = []

if not documents:

return results

for document in documents:

score = 0.0

for token in analyzed\_query:

tf = document.term\_frequency(token)

idf = self.inverse\_document\_frequency(token)

score += tf \* idf

results.append((document, score))

return sorted(results, key=lambda doc: doc[1], reverse=True)

Using TF-IDF i was able to quantify words in as all sets of documents hence creating a score that signifies importance of document and corpus

Tf -idf = Term Frequency \* Inverse Document Frequency

Term Frequency = count of terms in a document/number of words in a document

In this case, Using Counter to calculate the number of times a word appears inside a single document.

def analyze(self):

self.term\_frequencies = Counter(cleaning\_query(self.text))

def term\_frequency(self, term):

return self.term\_frequencies.get(term, 0)

Inverse Document frequency = inverse of document frequency

where, Document frequency =this is the measure of how important a document is a complete set of corpuses. Which means this is the number of documents in which the word is present

Formula:

Df = occurrences of a word in x number of documents

Code

def document\_frequency(self, token):

return len(self.index.get(token, set()))

Inverse frequency = Number of documents/document frequency,

Since we have a large corpus 10,000 the IDF value gets bigger, to dampen the effect, we take the log of IDF:

idf(t) = log (N/ (df + 1))

tf-idf(t, d) = tf(t, d) \* log (N/ (df + 1))

Where: T= term/token

D = document

N = corpus size

Execution on code

def inverse\_document\_frequency(self, token):

return math.log10(len(self.documents) / self.document\_frequency(token))

def rank(self, analyzed\_query, documents):

results = []

if not documents:

return results

for document in documents:

score = 0.0

for token in analyzed\_query:

tf = document.term\_frequency(token)

idf = self.inverse\_document\_frequency(token)

score += tf \* idf

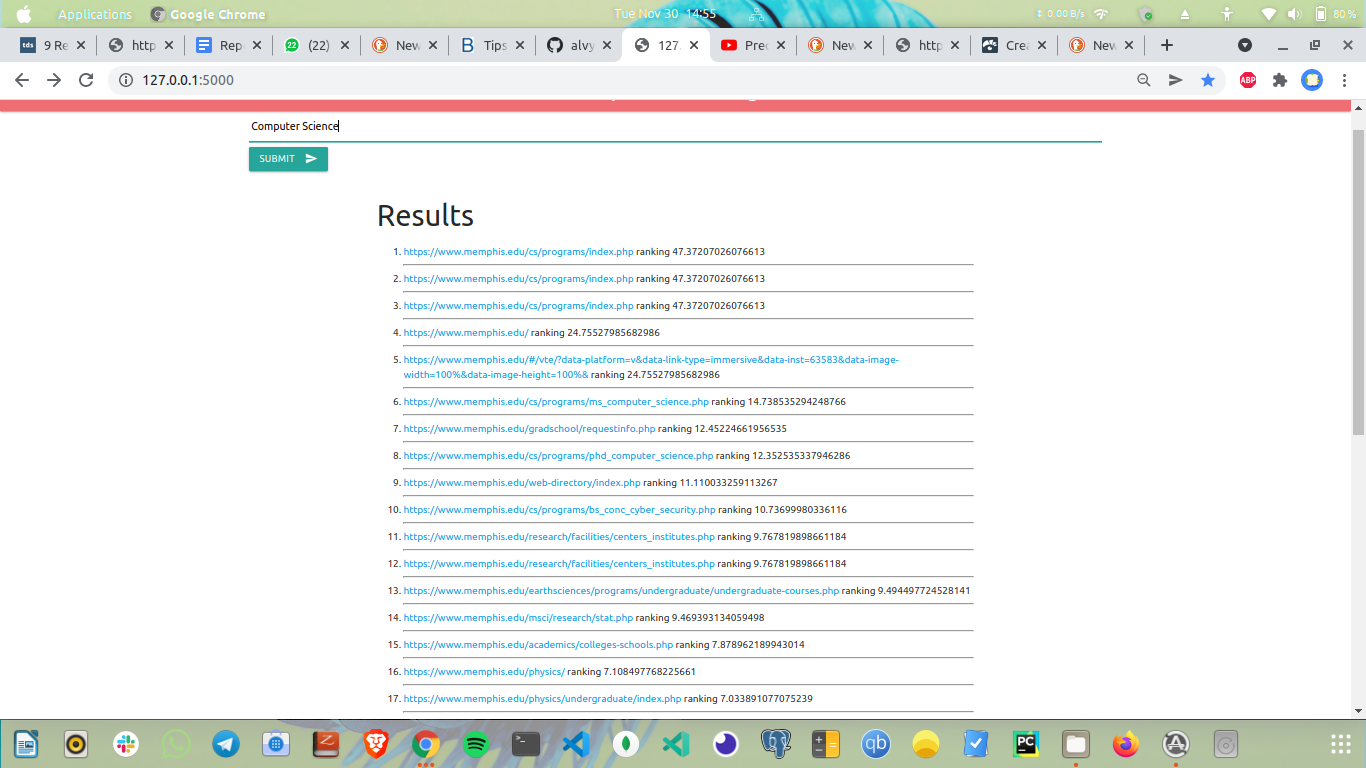
results.append((document, score))

return sorted(results, key=lambda doc: doc[1], reverse=True)

### 

# RESULTS

* q1: Computer Science



Total results = 31

Relevant and retrieved =13

Irrelevant and retrieved = 18

Precision = Relevant items retrieved/retrieved items

13/ (13+18) = 0.290323

Recall = relevant items retrieved / relevant items

=13/ (13+31)

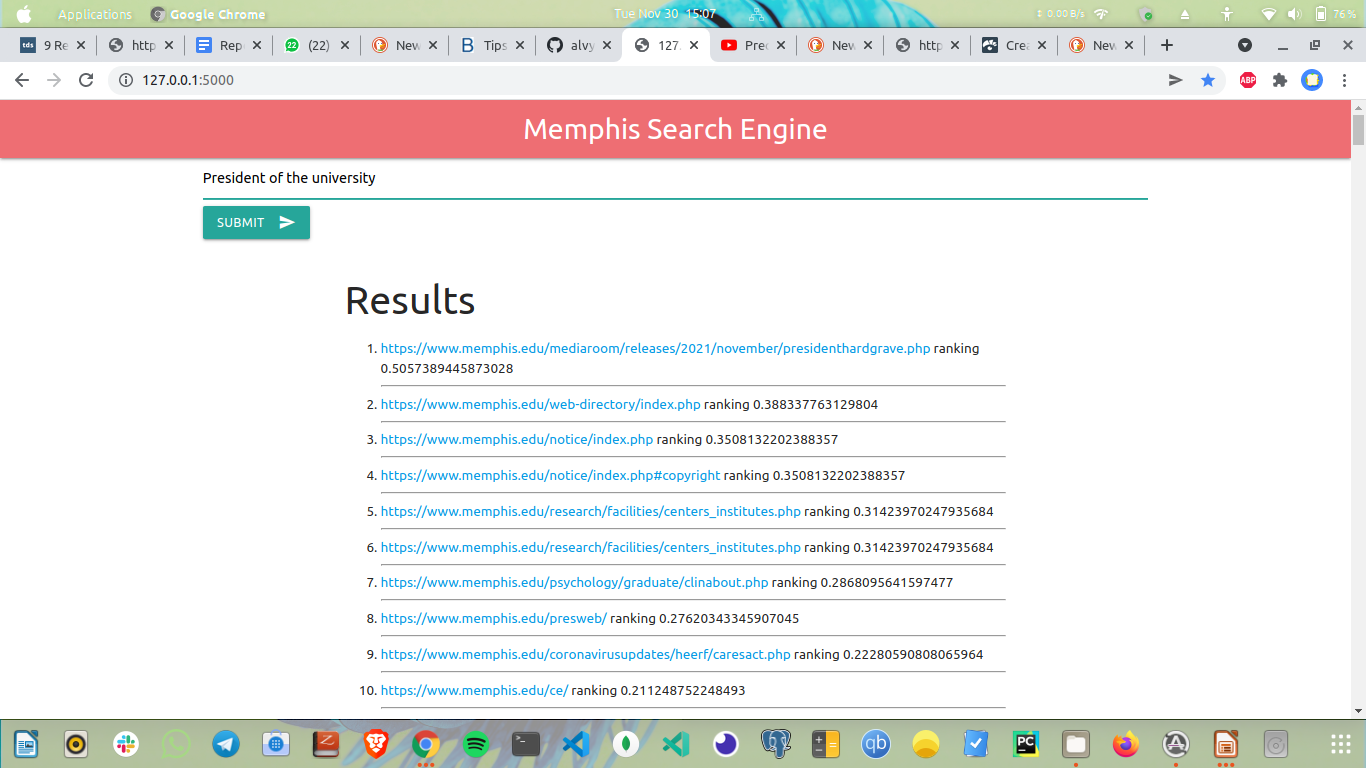
=0.2955

F1 score= 2(1/ (0.2903

+0.2955))

=3.41400

* q2: President of the university



Total retrieved = 519

Relevant = 2

Irrelevant and retrieved = 517

Precision = 2/519

= 0.0038

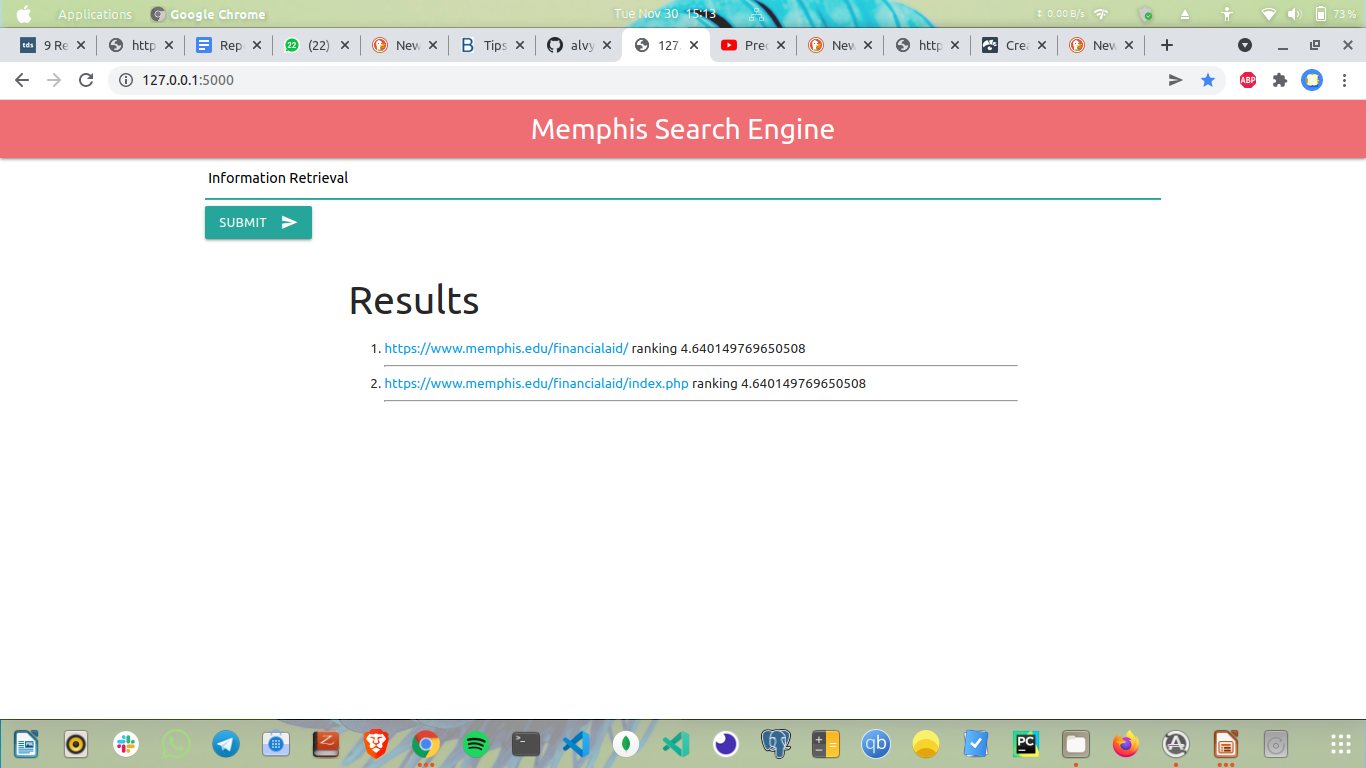
Recall =2(2+519)

=0.003838

F1 score = 2(1/ (0.0038+0.0038))

=263.1578

* q3: Information Retrieval



Retrieved =5

Relevant = 1

Irrelevant and retrieved = 4

P = TP/(TP+FP)

R = TP/(TP+FN)

Precision = 2/ (2+5)

=2/5

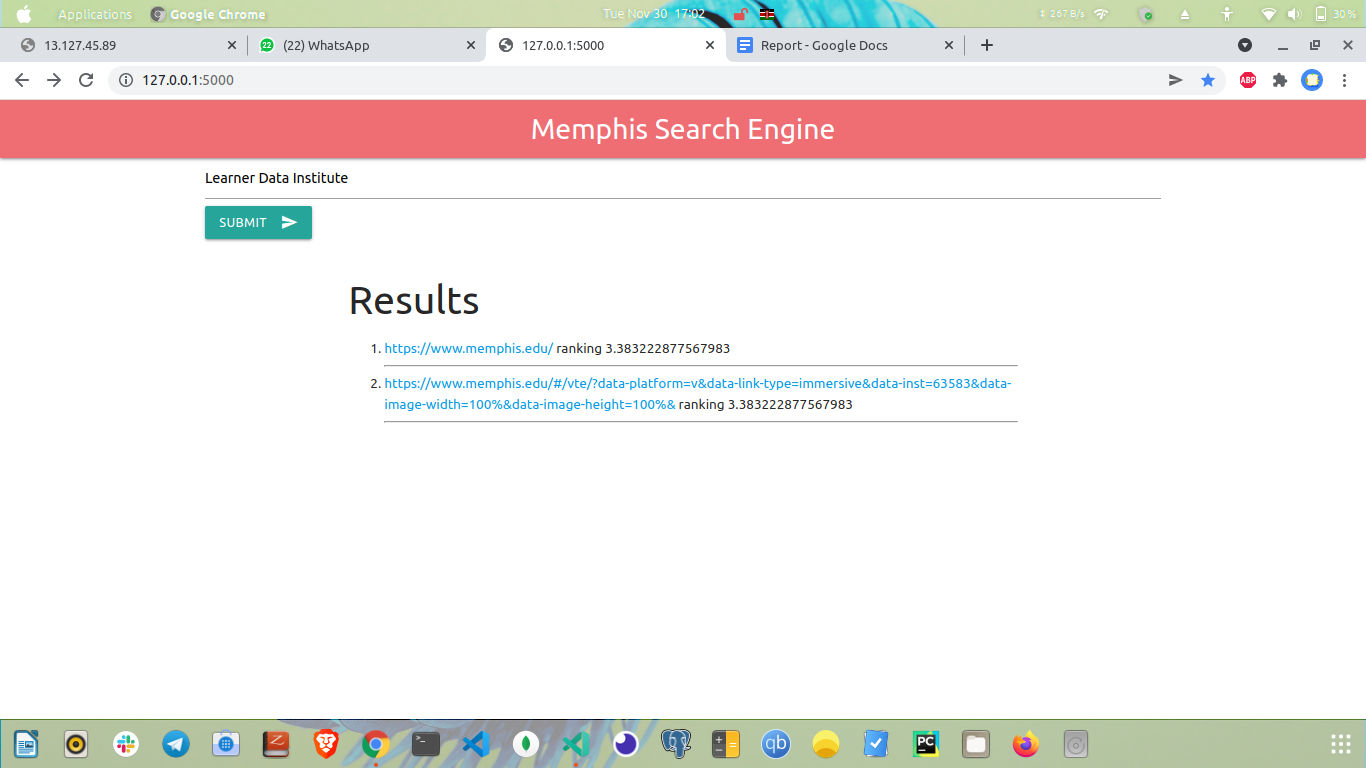
Recall=2 / (2+5)

=2/7

F1 score = 2(1/ (2/5+1/7))

=2.9167

* q4: Learner Data Institute



Retrieved = 2

Relevant = 0

Irrelevant and retrieved = 2

P = TP/(TP+FP)

Precision = 0/2

=0/2=0

R = TP/(TP+FN)

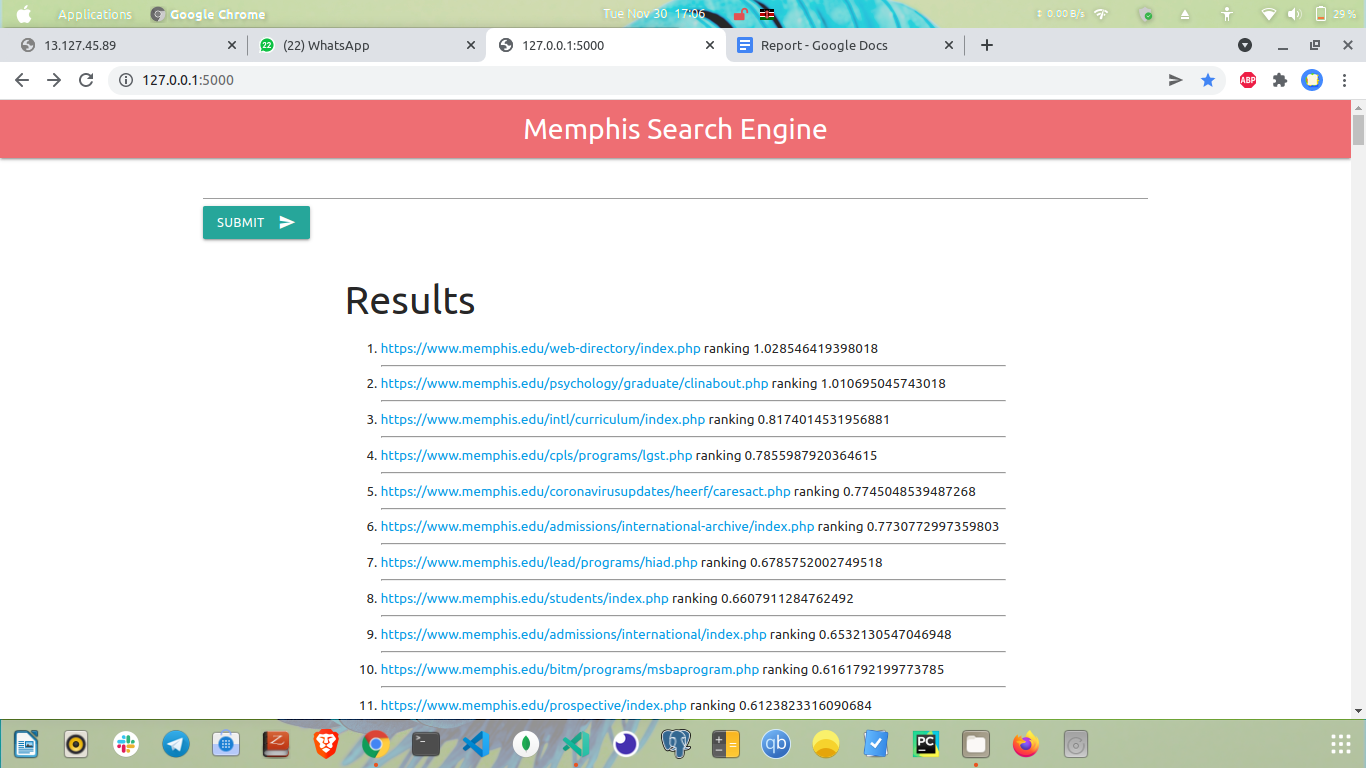
Recall = 0(0+2)

= 0

F1 score = 2(1/ (0/2+0/2))

= 0

* q5: International student office



Retrieved =337

Relevant = 6

P = TP/(TP+FP)

Precision = 6/337

=6/337

R = TP/(TP+FN)

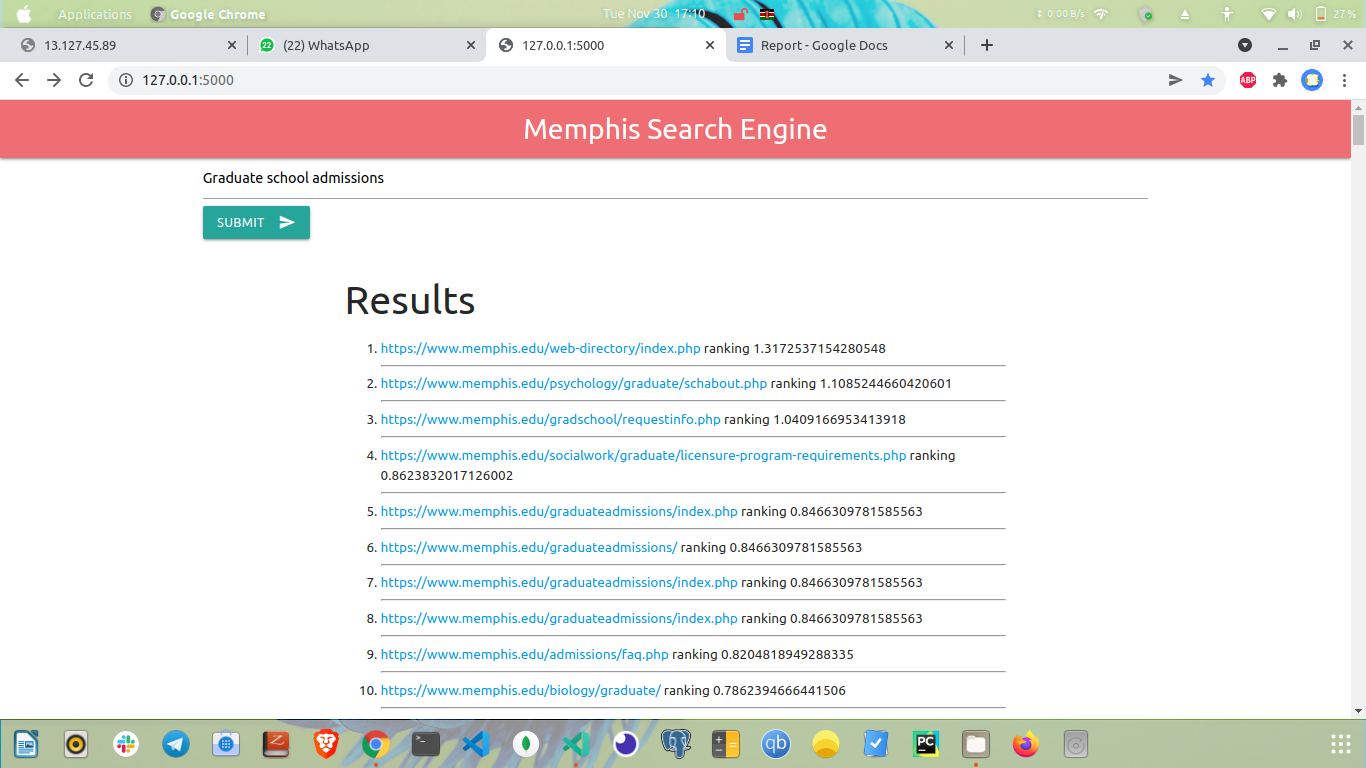
Recall = 6(6+337)

= 6/343

F1 score = 2(1/ (6/337+6/343))

= 56.6622

* q6: Graduate school admissions



Retrieved = 316

Relevant = 71

P = TP/(TP+FP)

=71/316

R = TP/(TP+FN)

=71/ (71+316)

=71/387

F1 score = 2(1/ (71/316+71/387))

=4.9002

* q7: What is the mascot of the University of Memphis?Graphical user interface, text, application

  Description automatically generated

retrieved = 1

relevant and retrieved =1

P = TP/(TP+FP)

Precision = 1/(1+1)

1/2

R = TP/(TP+FN)

Recall = 1/(1+1)

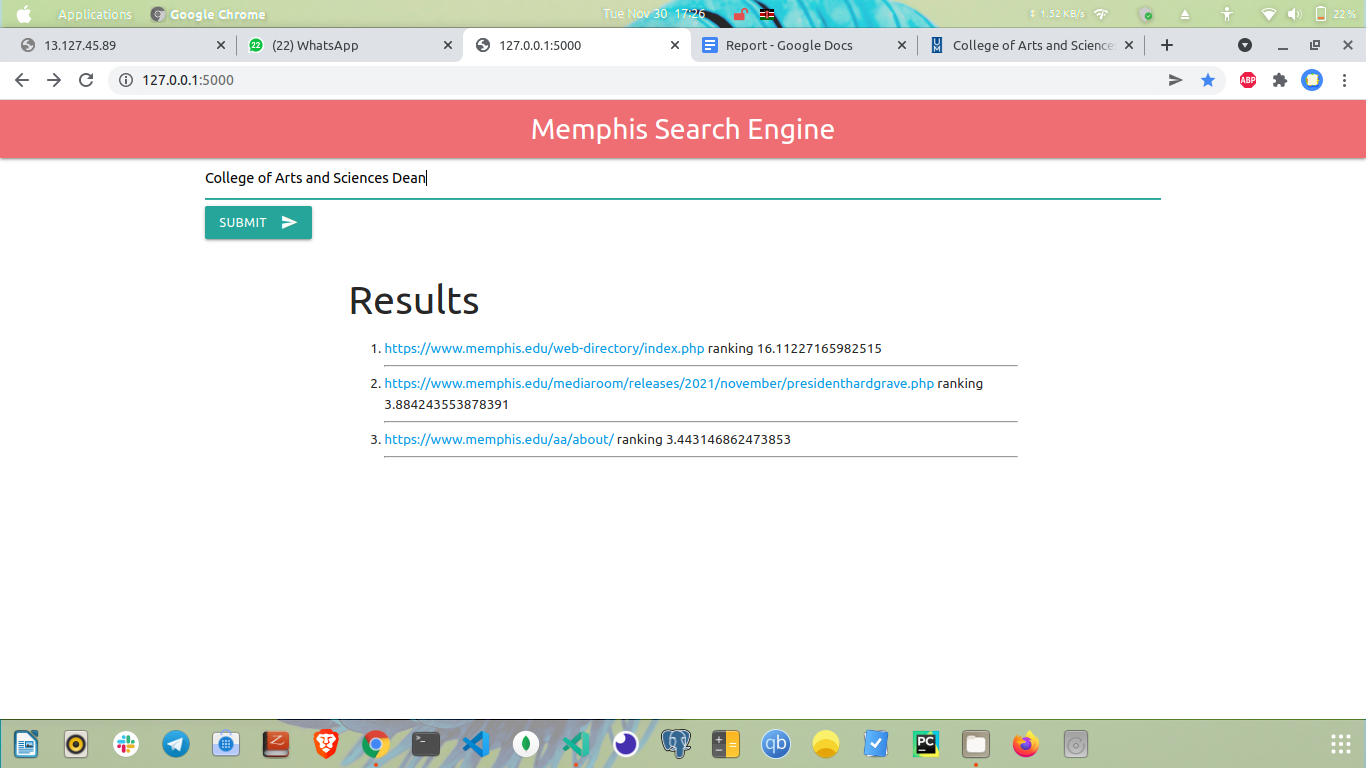
=1/2

F1 score = 2(1/(1/2+1/2))

=2(1/1/2)

=4

* q8: College of Arts and Sciences Dean



Retrieved = 3

Relevant = 2

P = TP/(TP+FP)

Precision = 2/3

=2/3

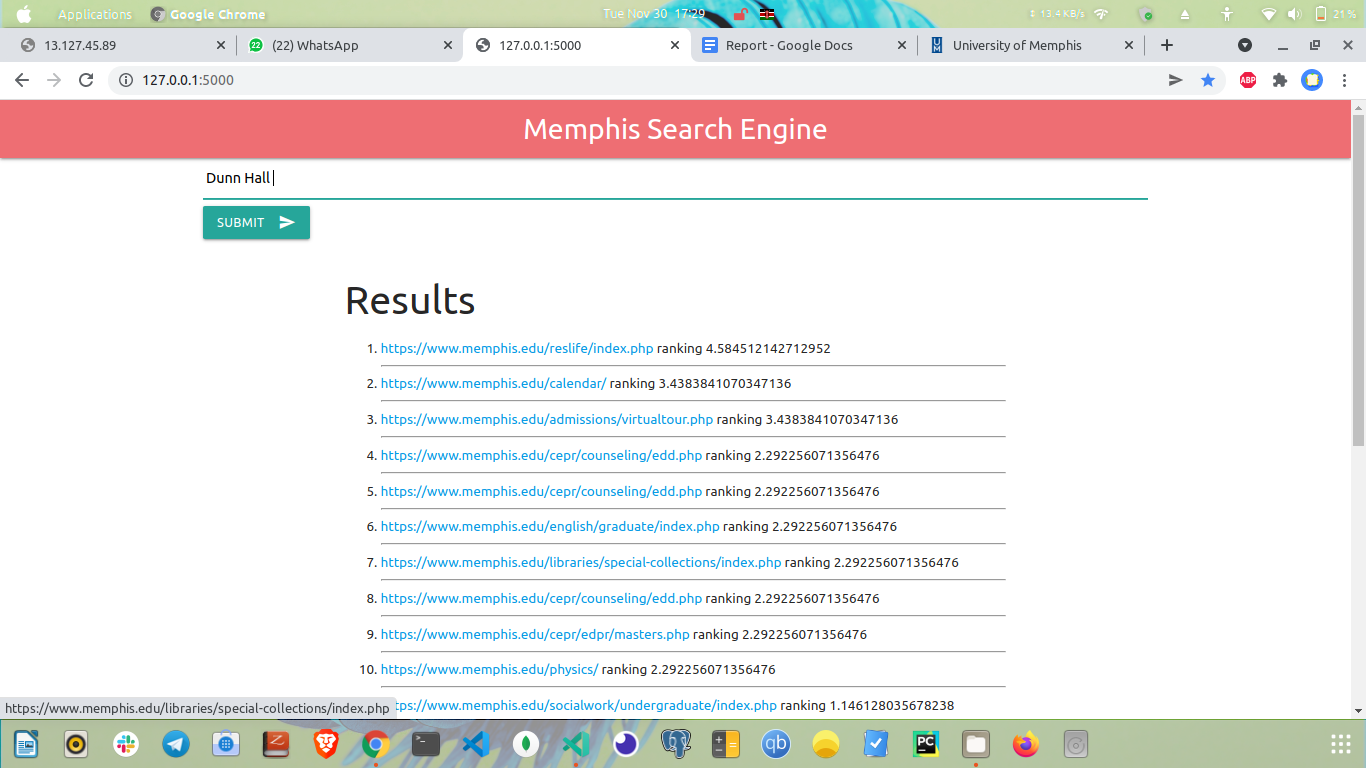
R = TP/(TP+FN)

=2(2+3)

=2/5

F1 score =2(1/ (2/5+2/3))

=1.875

* q9: Dunn Hall 

Retrieved = 27

Relevant = 9

irrelevant and retrieved = 18

P = TP/(TP+FP)

precision=9/27

0.3333

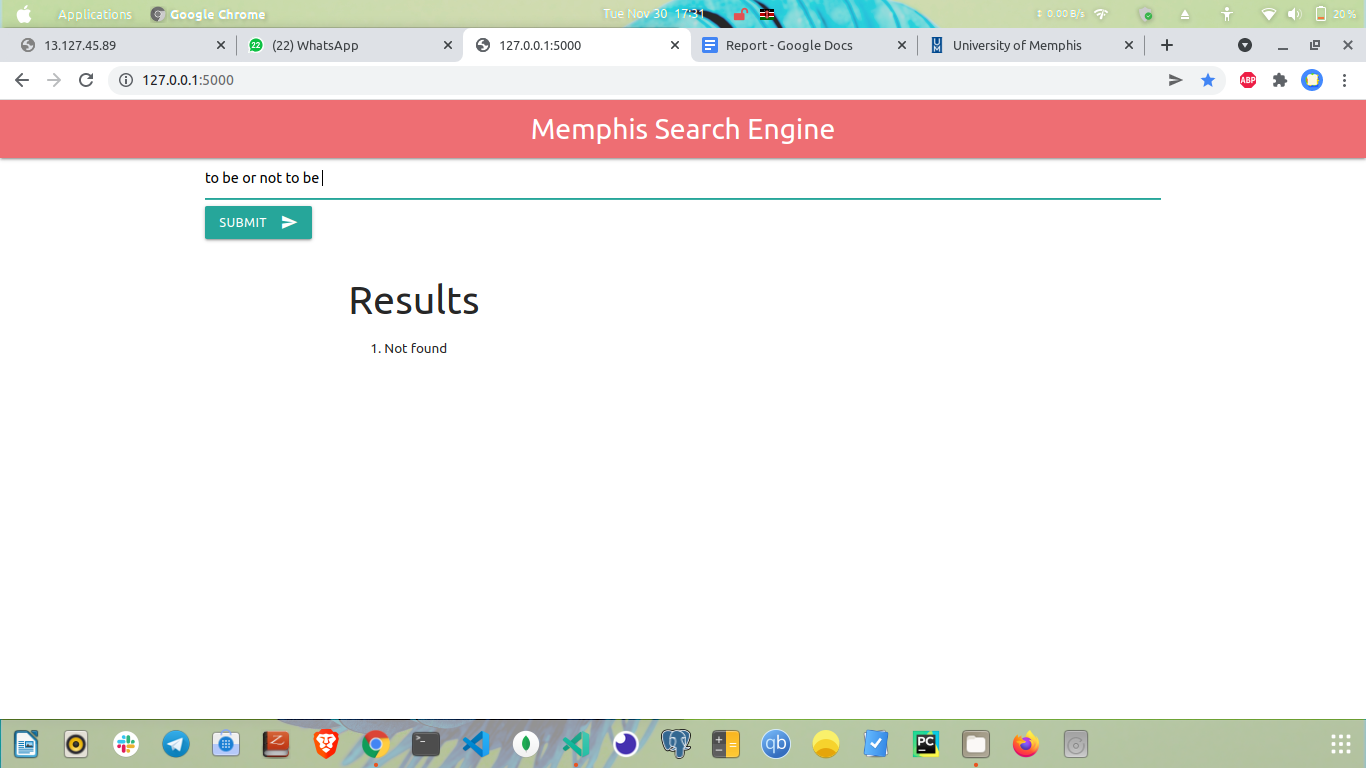
R = TP/(TP+FN)

Recall = tp(tp+fn)

=9(9+27)

=0.25

* q10: to be or not to be



Nothing was returned because all the words were stopped words

P = TP/(TP+FP)

Precision = 0

R = TP/(TP+FN)

Recall = 0

F1 score = 0

### Future work

1. Improving indexing speed.
2. Due to memory constraint not, all documents were able to be indexed, in future I am looking forward to new innovative ways to improve this.
3. There is more need to improve search accuracy.

# 

# Reference

1. beautiful-soup-4. (2021, November 27). Retrieved from beautiful-soup-4: https://beautiful-soup-4.readthedocs.io/en/latest/
2. ines, s. m. (2021, november 26). explosion. Retrieved from github.com: https://github.com/explosion/spaCy/blob/master/spacy/lang/en/stop\_words.py
3. logging — Logging facility for Python. (2021, November 26). Retrieved from logging — Logging facility for Python: https://docs.python.org/3/library/logging.html?highlight=logging#module-logging
4. Natural Language Toolki. (2021, November 26). Retrieved from Natural Language Toolki: https://www.nltk.org/
5. Regular expression operations. (2021, November 23). Retrieved from Regular expression operations: https://docs.python.org/3/library/re.html
6. urllib — URL handling modules. (2021, November 27). Retrieved from python documentation: https://docs.python.org/3/library/urllib.html#module-urllib