May, 2021

IN104 Project Report: Covid 19 Search Engine:

Dr.LOOK

by : Youssef BEN CHEIKH, Yassine OJ.
supervised by : Ms. Jessica Lopez Espejel.

Summary

1 Introduction

1.1 Context

An important number of health paper is published every year. However, due to the pandemic, the last year have known more activity. More efforts were put in order to tackle Covid-19. For instance :

1.In 2020, the number of submissions to Elsevier's journals increased by 58 % between February and May compared to the same period in 2019.

2.The number of health articles increased by 92% in 2020, where scientists published more than 100,000 articles about the Covid-19.

3.According to the EMC Digital Universe with Research and Analysis by IDC1 there was an enormous growth in the global healthcare data between 2013 and 2020.

1.2 Problem

The more data we have, the more it becomes challenging to find articles related to a field of interest. Which will waste time and slow scientific achievements.

1.3 Approach

There's no better solution than delegating computers to do the search for you. However we need a more sophisticated tool, adapted with the scientific articles. Here comes the idea of designing a search engine specialized in scientific papers to help researchers and scientists find articles related to their field of work easier and faster.

2 Implementation Environment

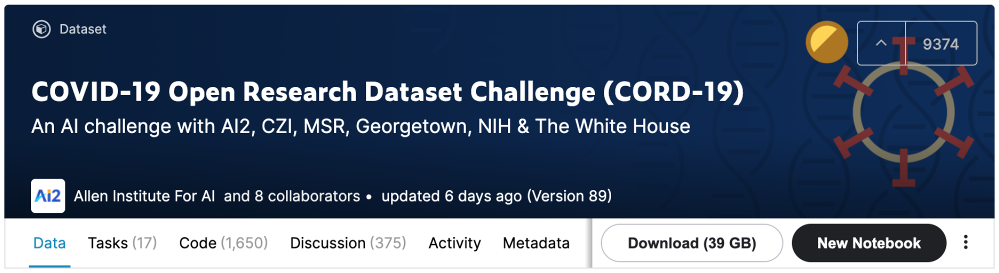
2.1 Tools

Dr.LOOK is based on the following tools :

- Python ≥ 3.6
- Natural Language Toolkit : a leading platform for building Python programs to work with human language data.
- Whoosh : a fast, pure Python search engine library.

2.2 Dataset

The data set provided for this project is of 147.047 health article in json format, a subset of COVID-19 Open Research Dataset (CORD-19)] that was proposed by the White House and research groups. It consists of more than 33 GB of information that includes target tablets, ready-to-use embeddings, and json articles.


Dataset
COVID-19 Open Research Dataset Challenge (CORD-19)
An AI challenge with AI2, CZI, MSR, Georgetown, NIH & The White House
9374
Allen Institute For AI and 8 collaborators • updated 6 days ago (Version 89)
Data
Tasks (17)
Code (1,650)
Discussion (375)
Activity
Metadata
Download (39 GB)
New Notebook

3 Classes and Methods

3.1 ExtractData class

At first, we need to load the data. We use read_json(path_file) to read a json file :

```
def read_json(path_file):
   """It reads a json document
   :return: json data
   :rtype dic_data: dictionary """
   with open(path_file) as json_file:
       dic_data = json.load(json_file)
       return dic_data


```

The return type is a dictionary. Based on that, we design the Extract data methods, to extract : paper id, the title and the body text.

```
  class ExtractData:
   def __init__(self, data):
       self.data = data
   def get_paper_id(self):
               return self.data['paper_id']
   def get_title(self):
              return self.data['metadata']['title'].lower()
   def get_text(self):
       L = self.data['body_text']
       text = ""
       for dic in L :
           text += dic['text']
       return text

```

Ces méthodes sont définies dans le fichier "extract_data.py" et utilisées dans "makecleandocs.py".

3.2 Pre-processing Class

```
  class PreprocessData:

   def __init__(self, text):
       self.text = text

```

After the extraction we need to get the data ready to be processed. For that we need to normalize the text first : lowercase the text, remove punctuation, special characters, numbers and english stopwords(such as ”the”,”an”,”in”,..). For the lowercasing it’s quite simple :

```
    def convert_lowercase(self):
          return self.text.lower()

```

For other methods we use the word-tokenizing technique to have a list of the text words, then we join only the words we want. We use this to remove punctuation. We can get the punctuation marks with string.punctuation : ”#$%&’()*+, -./:;¡=¿?@ [ ] ˆ_{ | } ˜

```
 def remove_punctuation(self, text):
         if isinstance(text, str):
             lst_words = [word for word in word_tokenize(text)
             if word not in string.punctuation]
         return ' '.join(lst_words)

```

In the same way we remove numbers, special characters and stop words. For numbers we use a function is_number(s) which returns True or False. We check if a character is special with character.isalnum()→False if character is special, we omet also the spaces with the special characters to remove the extra spaces.
For the stop words, we get them thanks to ntlk,

```
  from nltk.corpus import stopwords

```

Finally we lematize : we get the variant forms of a word to the original word (was, is, were, → verb to be). nltk provides a Lemmatizer that we can use :

```
from nltk.stem import WordNetLemmatizer
def lemmatization_text(self, text):
                if isinstance(text, str):
             lst_words = [self.lemmatizer.lemmatize(word)
             for word in word_tokenize(text)]
         return ' '.join(lst_words)

```

Those methods are implemented in ”preprocess_data.py” and used in ”makecleandocs.py”. In ”makecleandocs.py” we check the language of each article, so we can remove the non english ones. we use detect from the module langdetect, detect(text) return the string ”en” if text is in english.

```
from langdetect import detect
```

Clean docs look like :
stressinduced cell activation nos2 activity primarily regulated transcriptionally commonly induced bacterial product proinflammatory cytokine inflammatory disease respiratory tract asthma acute respiratory distress syndrome ards bronchiectasis commonly characterized increased expression nos2 within respiratory epithelial inflammatoryimmune cell markedly elevated local production no presumably additional host defense mechanism bacterial viral infection drawback excessive no production accelerated metabolism family potentially harmful reactive nitrogen specie rn including peroxynitrite onoo nitrogen dioxide no2 especially presence phagocytegenerated oxidant formation rn thought prime reason no many case contribute etiology inflammatory lung disease 456 despite extensive research proinflammatory antiinflammatory action no overall contribution no inflammatory .

3.3 Indexation

In the remaining part of work, we will rely mostly on whoosh.

"Whoosh is a library of classes and functions for indexing text and then searching the index. It allows you to develop custom search engines for your content. For example, if you were creating blogging software, you could use Whoosh to add a search function to allow users to search blog entries."(whoosh.readthedocs.io) Now, we have to create an index and add the documents to it. We use the function :

```
def create_index(save_index_folder, index_name,
num_docs_index, lst_articlesIndex, schema)
```

where save_index_folder is the path where to save the index, index_name is the name of the index, num_docs_index is the number of documents to index and lst_articlesIndex is the list of the documents paths.
The schema specifies the fields of documents in an index. Each document can have multiple fields, such as title, content, url, date, etc. Here we have path and content :

```
Schemah = Schema(path=TEXT(stored=True),
content=TEXT(analyzer=StemmingAnalyzer()))
```

path=TEXT(stored=True) specifies that the text should be stored in the index. content=TEXT(analyzer=StemmingAnalyzer()) means that it's the content which should be analysed, using StemmingAnalyzer(). Stemming is a heuristic process of removing suffixes (and sometimes prefixes) from words to arrive (hopefully, most of the time) at the base word. It allows the user to find documents without worrying about word forms but can sometimes incorrectly conflate words or change the meaning of a word by removing suffixes.

The function checks first if the index is already created, if not it creates and returns it, otherwise it just returns it.

```
 if not exists_in(save_index_folder, index_name):
  .
  .
  .
 ix = open_dir(save_index_folder, index_name)
 return ix

```

With the method get_content we get the content of each file. And using ix.writer.add_document(path=path_name_file_index, content=article_content) in a loop we add the documents to the index.

sys.stdout.flush() is used to flush the buffer(ia region of physical memory storage used to temporarily store data while it is being moved from one place to another.) every time we process 10000 document. Python's standard out is buffered. This means that it collects some data before it is written to standard, and with flushing we make sure we don't have any memory issues.

For the project I took :

```
    num_docs_index = 5000
    number_docs_result_search = 10

```

3.4 Search

For the search we use whoosh Qparser and the index.searcher().

```
searcher = ix.searcher()
parser_query = QueryParser("content", schema=Schemah)
q = parser_query.parse(sys.argv[3])
results = searcher.search(q, limit=
number_docs_result_search)

```

QueryParser("content", schema=Schemah) specifies that we want to look in the field : content, defined with Schemah. q = parser_query.parse(sys.argv[3]) receives the keywords and returns the search request. The keywords must be inserted in command line after the two paths, the one of the documents and the other where the index is saved. searcher.search does the search according to the request and returns the results. Like that we get our search results in a list ordered from the most related to the less. Now we need just to show display them.

How to run the search engine :
Cleaning docs and getting them ready to index:

```
 python makecleandocs.py /Users/Path_to_json_data_base/
 /Users/Path_where_to_save_clean_docs/

```

Index if not indexed yet and search:

```
python indexandsearch.py /Users/Path_to_Clean_Docs/
/Users/Path_where_to_save_index/ "keywords you're looking
for"


```

3.5 Spelling Correction

I added this feature to the search engine so it can suggest three corrections for every misspelled word from the keywords. We build a corrector based on the articles, The advantage of using the contents of an index field is that when you are spell checking queries on that index, the suggestions are tailored to the contents of the index. The disadvantage is that if the indexed documents contain spelling errors, then the spelling suggestions will also be erroneous. But we will trust the Academic Articles.

```
 corrector = searcher.corrector("content")

```

We check the spelling of the query, if it is mistaken, we will ask the corrector for some suggestions :

```
corrected = searcher.correct_query(q, sys.argv[3])
    if corrected.query != q:
        mistyped_words = word_tokenize(sys.argv[3])
        for mistyped_word in mistyped_words:
            Listecorr = corrector.suggest(mistyped_word,
            limit=3)
            print("Did you mean",", ".join(Listecorr),
            "instead of", mistyped_word,"?")

```

As an example : for viruso it suggests virus virusin viruses.

4 Conclusions

- Some of Whoosh's features include:

Pure-Python. No compilation or binary packages needed without mysterious crashes.

Fielded indexing and search.

Fast indexing and retrieval.

Powerful query language.

Pure Python spell-checker.

- OPP is very helpful. It makes the code diagnostic and maintenance easier. It secured the data and makes you more productive.

- GITHUB is a powerful tool for team work programming with all the features it provides.

- To save more time we can make the engine search in real time, while you're typing.