CoviText: Motor de Pesquisa da Literatura Médica sobre a COVID-19

CoviText: Search Engine of the Medical Literature about COVID-19

CoviText: Motor de Búsqueda de Literatura Médica en COVID-19

Jedson Gabriel Ferreira de Paula[1], Rômulo César Silva[2]

1 Discente de Ciência da Computação na Universidade Estadual do Oeste do Paraná - Campus de Foz do Iguaçu.
2 Professor adjunto de Ciência da Computação na Universidade Estadual do Oeste do Paraná - Campus de Foz do Iguaçu.

Autor correspondente: Jedson Gabriel Ferreira de Paula
E-mail: jedson_gabriel@hotmail.com

Resumo

Nos primeiros meses de 2020, a pandemia de COVID-19 afetou vários países, incluindo o Brasil. Devido à possibilidade de colapso nos sistemas de saúde e prejuízos econômicos, há um número crescente de pesquisas relacionadas, gerando um grande volume de artigos científicos, a exemplo da base CORD-19 (87,5 GB, ano base: 2022), disponibilizada pela Kaggle. Devido à rápida aceleração no crescimento da literatura sobre o novo coronavírus, dificultando o acompanhamento da comunidade de pesquisa médica, é necessário buscar formas de navegação e consulta sobre o que já é conhecido sobre o vírus Sars-Cov-2 e sua doença correlata. No intuito de facilitar esse processo, este trabalho apresenta a ferramenta CoviText: um motor de pesquisa da literatura médica sobre a COVID-19, desenvolvido utilizando técnicas de mineração de texto e aprendizado de máquina.

Descritores: COVID-19; Aprendizado de Máquina; Mineração de Dados

Abstract
In the first months of 2020, the COVID-19 pandemic has severely affected several countries, including Brazil. Due to the possibility of collapse in health systems and economic losses, there is an increasing number of related researches, generating a large volume of scholarly articles, such as the CORD-19 dataset (87.5 GB, base year: 2022), available at the Kaggle website. Due to the rapid acceleration in new coronavirus literature, making it difficult for the medical research community to keep up, it is necessary to find ways of browsing and consulting what is already known about the Sars-Cov-2 virus and its related disease. In order to facilitate this process, this work presents the CoviText tool: a search engine for the medical literature on COVID-19, developed using text mining and machine learning techniques.
Keywords: COVID-19; Machine Learning; Data Mining

Resumen
En los primeros meses de 2020, la pandemia de COVID-19 afectó a varios países, incluso Brasil. Debido a la posibilidad de colapso de los sistemas de salud y pérdidas económicas, cada vez hay más investigaciones relacionadas, generando un gran volumen de artículos científicos, como la base de datos CORD-19 (87,5 GB, año base: 2022), puesta a disposición por el Kaggle. Debido a la rápida aceleración en el crecimiento de la literatura sobre el nuevo coronavirus, lo que dificulta el seguimiento de la comunidad de investigación médica, es necesario buscar formas de navegar y consultar lo que ya se sabe sobre el virus Sars-Cov-2 y sus enfermedad relacionada. Para facilitar este proceso, este trabajo presenta la herramienta CoviText: un motor de búsqueda de literatura médica sobre COVID-19, desarrollado utilizando técnicas de minería de texto y aprendizaje automático.
Descriptores: COVID-19; Aprendizaje automático; Minería de Dados

Introduction

As reported by the Brazilian Ministry of Health [(1)], COVID-19 is a disease caused by the coronavirus SARS-CoV-2 and displays a clinical status that goes from asymptomatic infections to severe respiratory condition. According to World Health

Organization (WHO) the majority of cases (around 80%) can display no symptoms, around 20% may need hospitalization as a consequence of respiratory complications and, of those, 5% will be required respiratory support machines as for respiratory failure. There was a large concern regarding the understanding of this virus given all information that was gathered as well as the knowledge of how to deal with the volume of infected people, so no essential health service got under too much load, and by doing so, causing loss of lives.

On March 12, 2020, the CORD-19 dataset was published on Kaggle(2) website, a popular Data Science community platform. In this dataset, there are over a million articles and papers about multiple aspects of the coronavirus and its impact on different areas. Along with CORD-19 dataset, there are tasks asking important questions about its influence on the world around and also on our bodies and mind. All the texts available are organized and stored in JSON files, also containing a lot of information on the papers' metadata.

Collecting the data in such organized fashion is a complex task itself and, as it stands, the dataset is ready for the application of data mining algorithms in order to get insights or even to offer tools for domain experts to manipulate and learn from.

A great effort carried out by the data scientists has been focused in understanding the trends of infection and mortality rates on all regions of the globe, applying time series analysis and forecasting. Although it’s an important concern in regard to maintaining the hospitals under capacity and avoiding disasters, there are other areas that should be considered.

In this work, we will discuss techniques used to break down the dataset into a machine learning friendly structure, and we present a tool developed to help extract information that can contribute to new insights.

Related Work

Deep learning techniques used in natural language processing (NLP) have been applied for biomedical text mining models. According to Lee *et al.* (3), directly applying state-of-the-art NLP methodologies to biomedical text mining has limitations, because they are usually trained and tested mainly on datasets containing general domain texts, it is difficult to estimate their performance on datasets containing biomedical texts. Also, the word distributions of general and biomedical corpora are quite different, which can often be a problem for biomedical text mining models.

Lee *et al.* (3) developed BioBERT, a domain-specific language representation model pre-trained on large-scale biomedical corpora, that significantly outperforms other models on the biomedical named entity recognition, biomedical relation extraction and biomedical question answering.

CovidBERT (4) is a biomedical relationship extraction model based on BERT that extracts new relationships between various biomedical entities. Specifically, it uses the pre-trained BioBERT model and train it on additional Covid-19 related corpus, including CORD-19 dataset.

Tian e Zhang (5) evaluated several deep learning models built on PubMedBERT, a pre-trained language model, with different strategies addressing the challenges of the multi-label classification task for automated topic annotation of COVID-19 literature. Multi-instance learning was used to deal with the large variation in the lengths of the articles, and focal loss function was used to address the imbalance in the distribution of different topics.

Methods

Text Mining

Text mining is the extraction process of useful information from text data. This entails the identification and extraction of patterns within the text using a plethora of techniques of statistical processing, machine learning and natural language processing.

Text mining can be used to discover new relationships not apparent in the data, in order to generate new hypothesis and automate the knowledge extraction from the data. Can be best used for text classification and trend detection.

One of the techniques studied in the text mining domain is the text encoding (or text embedding), as it’s an important concept needed for the further steps in this work. We can understand “embedding” as the vector representation of a text document as produced by a text transformation model.

The models that can be used, can have many levels of complexity, from a simple bag of words to a state-of-the-art machine learning model. They all have the same objective: extract the semantics of the text and encode it in a computational efficient format, namely a vector of N dimensions, being N of any arbitrary size or limited by the model implementation.

The CoviText Application

CoviText provide a simple, easy to use, and intuitive interface to search for and view information about the COVID-19 pandemic from an academic perspective. The application provides a search bar and, optionally, a set of parameters that the user can set in order to modify the search criteria. After that, the application displays a list of articles, and the user is able to get articles to view and extract information from.

Choice of Technologies

To develop an application, it’s necessary to browse through the countless technologies available now-a-days and choose a maintainable and scalable one. By doing so, we can ensure that the application will be able to run for a long time and offer a good user experience.

Embedding Model

In order to make it possible to relate the queried sentence to the database of articles, we use a machine learning algorithm to predict the sentence's relevance to the articles. Specifically, the BERT (6) model is used as the primary embedding model. This model can later be used as a baseline, in this application, so future works can compare the embedding quality against distilled and specialized models.

Database

The database used for the storage of the data is PostgreSQL (7), a RDBMS (Relational Database Management System) that offers complete SQL functionality and a big collection of configuration and tuning possibilities that make it able to search efficiently thousands of documents.

As the database does not support vector embeddings natively, it's necessary to add this functionality with an extension, there's one available called vector and was created by a team of researchers (8). Using that extension, it's possible to store, index and calculate the inverse cosine similarity between the embeddings inside the database.

Back-End Server

The Rust (9) language is used to code the back-end. It is a systems programming language focused on safe memory management and performance. Rust is a modern, statically typed, compiled language that is used to write high-performance, distributed systems and applications. Although there are other languages that can be used to write the back-end API, Rust offers a new approach by making it easier to scale horizontally (accepting more concurrent connections) if needed.

Also, the Rocket (10) library was used in the development of the back-end API middle-ware. It is a high-level, modern, zero-configuration and flexible web framework for Rust, that allows the writing of web applications in a way that is idiomatic for Rust.

Front-End Server

For the front-end API, an easy-to-use, modern, and responsive web framework was needed, and for that reason, Svelte was used (11). Svelte is a JavaScript framework designed to be used as a single page application. This framework showed good performance, was simple to iterate over, and that is crucial in getting the application into a usable state. To help style the application, a CSS framework called TailwindCSS (12) was used as it offers a lot of styling options, and it works with a different paradigm than the other frameworks, by focusing on the utility classes, we can create a myriad of compound designs and reuse components.

Populating the Data

Before populating the data, a schema was needed to encode the correct relationships needed for the application. The following Entity Relationship diagram shows the schema used to encode the data:

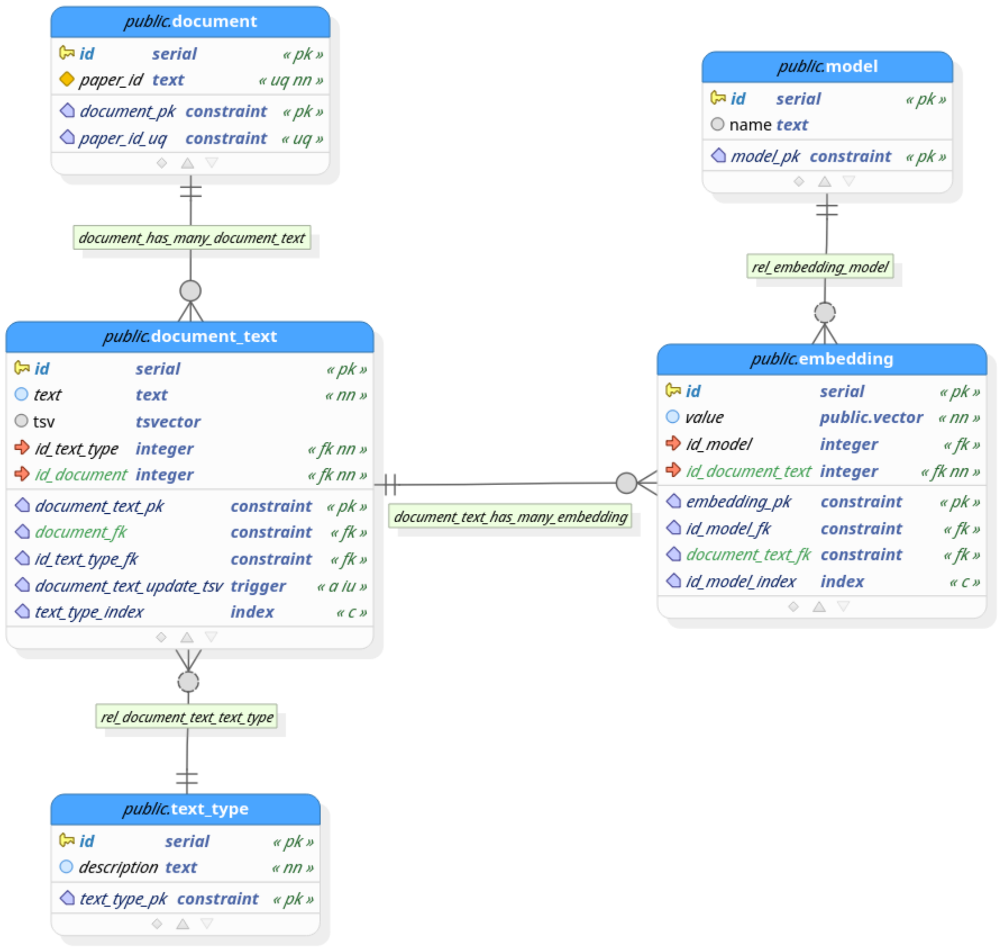

public.document
id serial « pk »
paper_id text « uq nn »
document_pk constraint « pk »
paper_id_uq constraint « uq »
document_has_many_document_text
public.model
id serial « pk »
name text
model_pk constraint « pk »
rel_embedding_model
public.document_text
id serial « pk »
text text « nn »
tsv tsvector
id_text_type integer « fk nn »
id_document integer « fk nn »
document_text_pk constraint « pk »
document_fk constraint « fk »
id_text_type_fk constraint « fk »
document_text_update_tsv trigger « a iu »
text_type_index index « c »
document_text_has_many_embedding
public.embedding
id serial « pk »
value public.vector « nn »
id_model integer « fk »
id_document_text integer « fk nn »
embedding_pk constraint « pk »
id_model_fk constraint « fk »
document_text_fk constraint « fk »
id_model_index index « c »
rel_document_text_text_type
public.text_type
id serial « pk »
description text « nn »
text_type_pk constraint « pk »

Figure 1 – Entity Relationship was modeled using an application called PgModeler [(13)].

Using the schema showed in Figure 1, it's possible to populate the database with the records from the CSV file that contains all metadata about the papers in the CORD-19 dataset. Also making it possible to have multiple embedding models, so the user can pick and choose between them, comparing results and evaluating their accuracy in the returned papers list.

CoviText Architecture

The Figure 2 shows the architecture used to build the application, that covers 3 (three) machines and 3 (three) code repositories.

The first being the development machine, where the CORD-19 dataset resides. A Python script was created in order to mine the text data from the metadata CSV containing the records of the articles present in the corpus. This script was responsible for fetching the data from disk and making a POST request to the insertion API of the server machine. The fetched text was preprocessed in order to remove anomalies or unusable records, e.g., empty titles or abstracts.

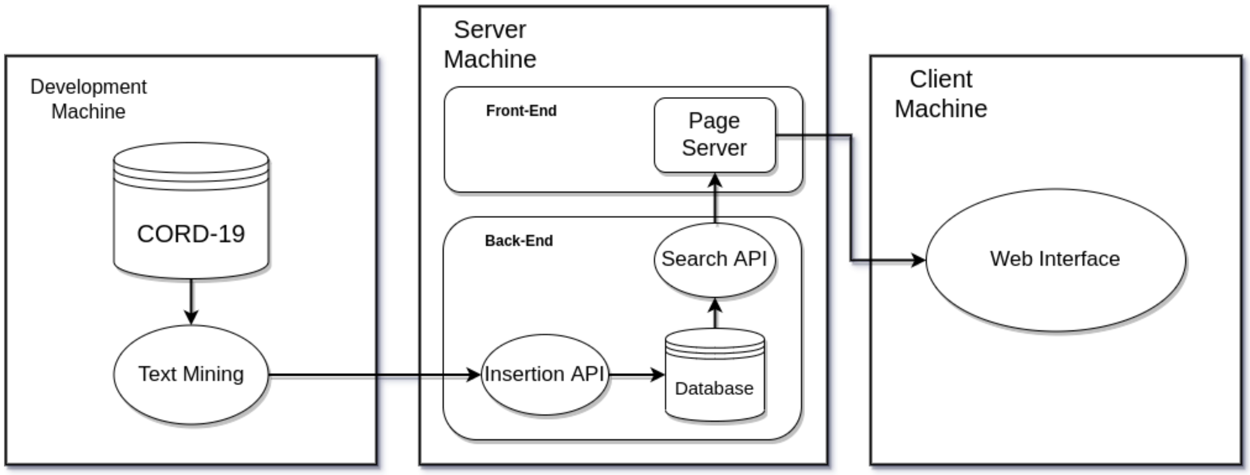


Figure 2 – Architecture of the CoviText

The second machine is responsible for the receiving, processing and storage of the data, as well as the serving of any query made. The receiving is taken care by the use of endpoints for the insertion API.

The endpoints accept a snippet of text and pass it to the models, so they can perform the encoding to an embedding and that is stored in the database alongside the snippet.

It's also available endpoints for a search API, where they encode the incoming text and execute a comparison between the embeddings stored in the database, returning the quantity of records requested in order of similarity, calculated with the inverse cosine function.

Inside the server machine it's hosted the code for the front-end pages, made so they can ease the use of the application, providing hints to what is searchable within the database. The search page implements API provided by the back-end and parses the results, having it be showed in a more presentable way.

Obstacles

As the query time is crucial for any search engine, some optimizations were needed in order to get a better response time from the back-end. One of those optimizations was to create a materialized view containing all embeddings from a single machine learning model, denormalizing some useful data from each text snippet and indexing it using the vector extension. Using that strategy, we got the response time unoptimized from around 2 (two) seconds to 200 (two hundred) milliseconds.

From the start of the development process to the current state, it's now clear how much the tooling for building fast and modern applications is at a really mature state. Good interconnection between all the technologies that were used was crucial in making this tool a possibility.

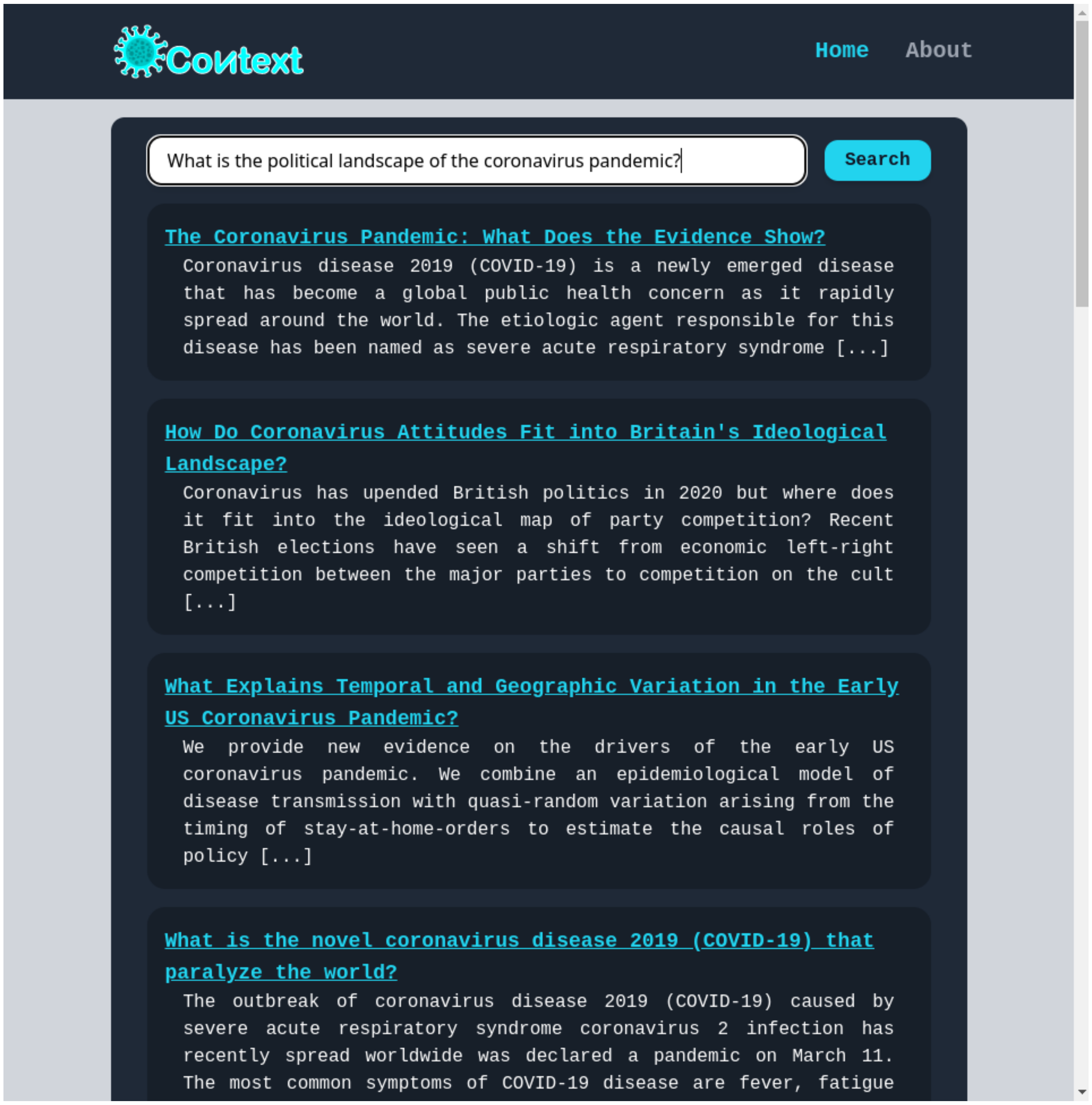
Some small obstacles are always present in any attempt to solve a problem, and this one is no exception. Those obstacles were the following:

- Volume of the data. The dataset is really large in file size, making up 87.5 GB of text. As the initial plan was to use all the full text, after attesting that the results from the paragraphs inside the papers did not make good results, it was quick to assert only to use the title and abstracts, both increasing the quality and decreasing the load on the back-end.
- Low latency queries. As state in the previous section, it is important for a search engine to have low latency, so the user can iterate over multiple queries until a relevant one is found. To achieve so, two optimizations were essential: Materialized Views; and Indexing. The former makes the data readily available for the most important query, that is, the search of relevant papers given a user input. The latter makes that the RDBMS does not have to fetch too much data into memory, speeding up the recall.

Results and Discussion

Search Interfaces are routines of any user web applications, being present in many applications and helping in navigation of content. Their design is different and simple, including an input text, which includes a search bar, and a list in cascade of title/description.

Not having much to diverge from this theme, we implemented something simple and direct that compromises the desired function. When styling, we searched a good contrast between text and background so that everything is readable, besides the use of an accessible color to people with some level of color-blindness. The page is rendered in the client's machine and the layout is presented in the figure 3.



What is the political landscape of the coronavirus pandemic?

The Coronavirus Pandemic: What Does the Evidence Show?
Coronavirus disease 2019 (COVID-19) is a newly emerged disease that has become a global public health concern as it rapidly spread around the world. The etiologic agent responsible for this disease has been named as severe acute respiratory syndrome [...]

How Do Coronavirus Attitudes Fit into Britain's Ideological Landscape?
Coronavirus has upended British politics in 2020 but where does it fit into the ideological map of party competition? Recent British elections have seen a shift from economic left-right competition between the major parties to competition on the cult [...]

What Explains Temporal and Geographic Variation in the Early US Coronavirus Pandemic?
We provide new evidence on the drivers of the early US coronavirus pandemic. We combine an epidemiological model of disease transmission with quasi-random variation arising from the timing of stay-at-home-orders to estimate the causal roles of policy [...]

What is the novel coronavirus disease 2019 (COVID-19) that paralyze the world?
The outbreak of coronavirus disease 2019 (COVID-19) caused by severe acute respiratory syndrome coronavirus 2 infection has recently spread worldwide was declared a pandemic on March 11. The most common symptoms of COVID-19 disease are fever, fatigue

Figure 3 – Application running on the development environment.

The interface provide a simple search functionality, where the desired query can be typed in the text input field and confirmed by either pressing the enter key on the keyboard or clicking the search button beside the search bar. The application is then going to create a request for the back-end API with some hard coded settings (that may become variable parameters in future versions). The request is going to be fulfilled and returned with a list of articles that show correlation with the queried text. The data is then showed in as a list in descending order of correlation. When any of the results are clicked, a modal with the full title and abstract text is shown (and for future versions there can be references pointing to where to access those resources).

The application is now green lighting (14), and any feedback or bug reports can be sent to any one of the authors by email.

CoviText as Framework

There is a different perspective to the work shown of the tool development, we can look at it as a framework of which it's possible to build search engines for a different corpus and use-case. Using one of the many corpora as a starting point, the text mining tools, the front and back end and the database can be applied and yield results that may justify the semantic search approach instead of a pure text matching one.

Conclusion

Since the start of the coronavirus pandemic, there was created a task force for the gathering and understanding of all aspects regarding the virus and the disease it causes. In an attempt to converge the effort, the AI2 team and White house created the CORD-19 dataset, in which there is a large volume of text data about the virus and it's aspects.

Using text mining techniques, the dataset was processed in order to create embeddings for each of the title and abstracts in the corpus. Those embeddings hold the semantics of the text they encode, so it was possible, by comparing each of those embeddings with each other, to check their similarity in terms of semantic content. We then harnessed this property by creating an application that generates an embedding as a result of a search done by a user and compares it to the database of embeddings created by the processing of the CORD-19 corpus.

The application was made using text mining libraries in Python, a back-end API written in Rust using the Rocket framework and for the front-end we used Svelte and TailwindCSS frameworks in the TypeScript language.

The implementation was filled with many challenges regarding the volume of the corpus and the criteria for a low latency response. To overcome those, more of the technology available was used. For the large volume, Dask offered a way to process data without loading all of it on working memory, and for the low latency, the vector extension gave us indexing options to reorganize the records, making the query run faster.

A possible next steps in the development of the application is the implementation of controls to pick and choose different embedding models, facilitating the comparisons between the results yielded. The extension used in the database to store and index the text encodings can be improved by using GPU acceleration, the project is open source so only the study and implementation is required.

References

1. O que é a Covid-19? [Internet]. Ministério da Saúde. [cited 2022 Jul 3]. Available from: https://www.gov.br/saude/pt-br/coronavirus/o-que-e-o-coronavirus

2. AI2, CZI, Georgetown, NIH, The White House. Covid-19 open research dataset challenge (CORD-19) [Internet]. [cited 2022 Jul 3]. Available from: https://www.kaggle.com/datasets/allen-institute-for-ai/CORD-19-research-challenge

3. Lee J, Yoon W, Kim S, Kim D, Kim S, So CH, et al. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. Wren J, editor. Bioinformatics [Internet]. 2019 Sep 10; Available from: https://academic.oup.com/bioinformatics/advance-article/doi/10.1093/bioinformatics/btz682/5566506

4. Hebbar S, Xie Y. CovidBERT-Biomedical Relation Extraction for Covid-19. FLAIRS [Internet]. 2021 Apr. 18 [cited 2022 Oct. 28];34. Available from: https://journals.flvc.org/FLAIRS/article/view/128488

5. Tian S, Zhang J. Multi-label topic classification for COVID-19 literature annotation using an ensemble model based on PubMedBERT [Internet]. 2021 Oct 29 [cited 2022 Oct 28]. Available from: https://www.biorxiv.org/content/10.1101/2021.10.26.465946v1.full

6. Devlin J, Chang MW, Lee K, Toutanova K. Bert: pre-training of deep bidirectional transformers for language understanding [Internet]. arXiv; 2019 [cited 2022 Jul 3]. Available from: http://arxiv.org/abs/1810.04805

7. Group PGD. Postgresql [Internet]. PostgreSQL. 2022 [cited 2022 Jul 3]. Available from: https://www.postgresql.org/

8. ankane. Vector: open-source vector similarity search for PostgreSQL [Internet]. PGXN: PostgreSQL Extension Network. [cited 2022 Jul 3]. Available from: https://pgxn.org/dist/vector/

9. Rust programming language [Internet]. [cited 2022 Jul 3]. Available from: https://www.rust-lang.org/

10. Rocket - simple, fast, type-safe web framework for Rust [Internet]. [cited 2022 Jul 3]. Available from: https://rocket.rs/

11. Svelte [Internet]. [cited 2022 Jul 3]. Available from: https://svelte.dev/

12. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. [Internet]. [cited 2022 Jul 3]. Available from: https://tailwindcss.com/

13. Silva RA e. Pgmodeler - PostgreSQL database modeler [Internet]. [cited 2022 Jul 3]. Available from: https://pgmodeler.io

14. CoviText [Internet]. [cited 2022 Jul 3]. Available from: https://covitext.jesuisjedi.com