Linguistic Engineering

Sars-engine : a collaborative framework for public-health SARS-CoV-2 prevention

Marcos Domínguez Velad

Mail : marcos.dvelad@alumnos.upm.es

Ilias Sarbout

Mail: ilias.sarbout@alumnos.upm.es

Year 2020-21

Contents

1 Introduction

The project consists of a question-answering system, in which the user can formulate queries and these are answered according to information extracted from selected texts.

In our case, we have chosen the current topic of the coronavirus as the domain of the application. This increases the variety of texts and information, so it is not trivial to handle them.

Our application presents a hybrid approach based, first of all, on the manual extraction of text from the World Health Organization website, making an automatic matching with flexible questions made by the user. Second, we have used a **corpus** of both: press and scientific texts, provided by the US government in Kaggle[1] [Wang et al, 2020]. Along with this corpus, we have applied the Infersent model from Facebook [Conneau et al, 2017] which, in addition to another **corpus of questions**, performs the matching between the user's questions and the corpus questions with a similarity metric to obtain the right answer. On the other hand, Infersent also makes an inference on the corpus of texts to look for answers to new questions.

We present the application flow and most important features in Section 2. Then, we explain the manual data extraction and automatic matching from texts, in addition to the most outstanding elements of this project such as the performance of Infersent in Section 3. As making the whole framework work is not easy, we provide a user manual in Section 4, pointing out the requirements to run it. Finally, we discuss some possible improvements for our framework in Section 5. In the Appendix the piece of text analyzed is given, and also a summary of the requirements.

If a user access to *sars-engine* through a web application or service, he may have access to a pre-established *search-engine-corpus* used by the server. Our applciation is built on a corpus we made, but our project also include the tools that allow anyone to build their own *search-engine-corpus*. The details on how to build it for are presented in section 3.4.

2 Application presentation

The main menu of the application is composed of the following options :

1. Ask a question.
2. Add a question and its answer.

The main option is the first one, "Ask a question", which allows the user to make a query, and will be detailed in the next section.

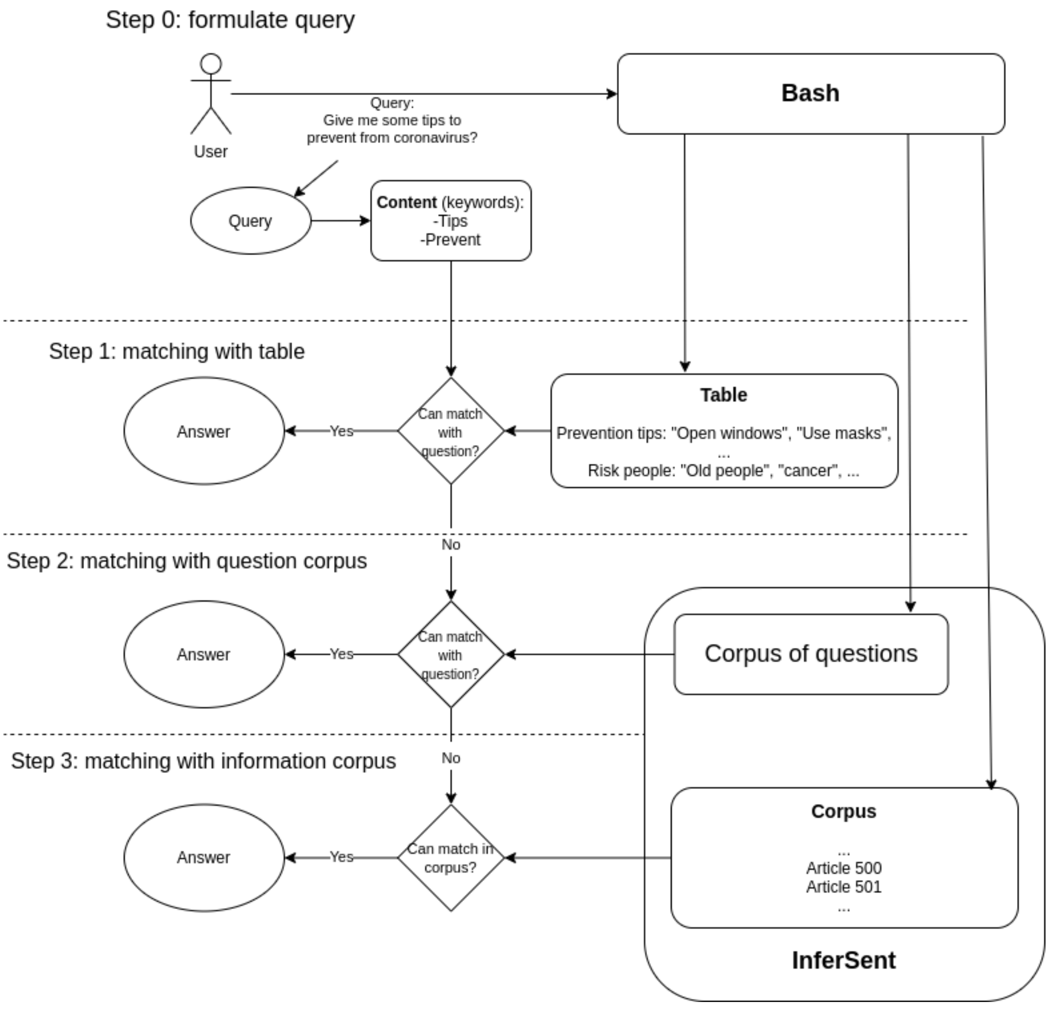
[1]You can find the corpus here:
https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge

The second query makes available the possibility of expand the *corpus of questions* with new questions and their answers.

Many other options are still in work (display a help text, licence informations, *etc.*).

2.1 Query processing (option 1.)

Figure 1 illustrates roughly the flow of execution of a query, starting from when the user enters it, until an answer is obtained.



Step 0: formulate query
User
Query: Give me some tips to prevent from coronavirus?
Bash
Query
Content (keywords): -Tips -Prevent
Step 1: matching with table
Table
Prevention tips: "Open windows", "Use masks", ...
Risk people: "Old people", "cancer", ...
Can match with question?
Yes
Answer
No
Step 2: matching with question corpus
Corpus of questions
Can match with question?
Yes
Answer
No
Step 3: matching with information corpus
Corpus
...
Article 500
Article 501
...
Can match in corpus?
Yes
Answer
InferSent

Figure 1: Flow chart of a query

Step 1

To process a query (option 1.), we first check if an analysis of the sentence allows to extract a categorical representation of the answer and of the main components of the query. We look in the table architecture if the answer can directly be found. This table is manually filled and expects to contain the more frequent asked questions and answers. In Section 3.1 we will describe this process in detail.

Step 2

Obviously, there is no way to make a complete table that would answer 100% of the possible queries. Moreover, filling this table requires many temporal investment and a daily update and maintenance. Therefore, When the answer is not found, we make use of the *corpus of questions*, another data structure that contains questions formulated in a natural way and their associated answers. We look for the most similar question in this corpus and ask the user if that is what he meant. In case it is, the answer is directly displayed.

Step 3

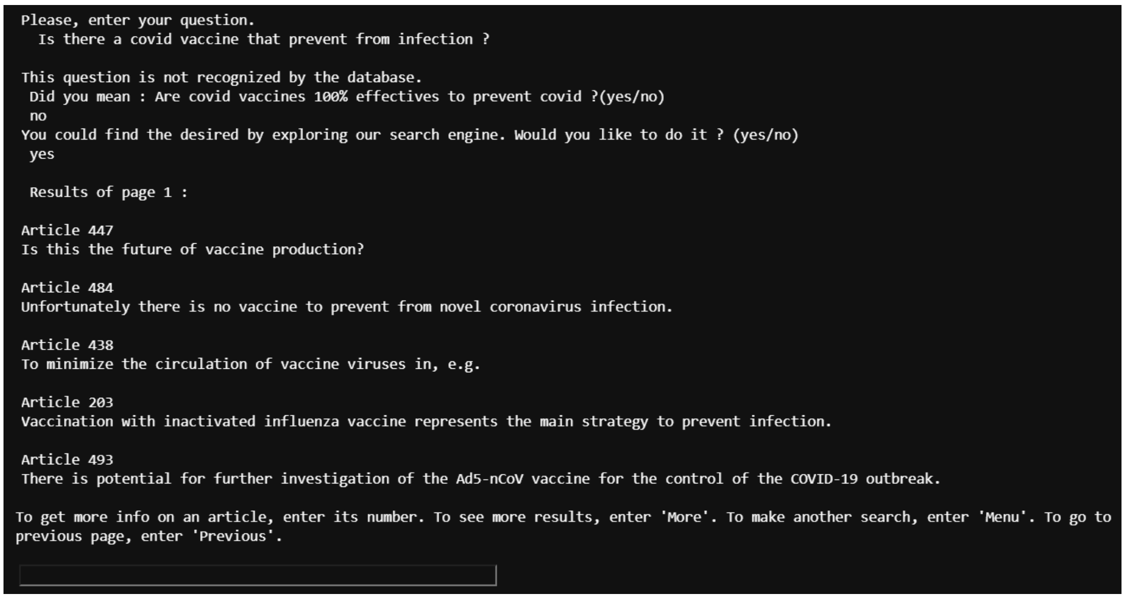
In case it isn't, the user can make use of the integrated search engine to find the desired information. The database of this search engine is composed of many pre-processed scientific and press articles about SARS-CoV-2. Results of similar content are printed and the user can navigate through pages and see sentences that may answer his question and their associated article numbers. The user can look onto the details of a specific articles (authors, abstract) and even read it.

Figure 2. show an example of query processing.

2.2 Enrich *corpus of questions* (option 2.)

The option number 2 allow the user to add a question and its answer in the *corpus of question*. This corpus is composed of two files : a panda dataframe containing questions and associated answers (*df_questions.joblib*) and the Infersent vectors representations of its questions (*embeddings_questions.joblib*)

When a new element is added, the two files are updated and saved again.



```
 Please, enter your question.
   Is there a covid vaccine that prevent from infection ?

 This question is not recognized by the database.
  Did you mean : Are covid vaccines 100% effectives to prevent covid ?(yes/no)
  no
 You could find the desired by exploring our search engine. Would you like to do it ? (yes/no)
  yes

  Results of page 1 :

 Article 447
 Is this the future of vaccine production?

 Article 484
 Unfortunately there is no vaccine to prevent from novel coronavirus infection.

 Article 438
 To minimize the circulation of vaccine viruses in, e.g.

 Article 203
 Vaccination with inactivated influenza vaccine represents the main strategy to prevent infection.

 Article 493
 There is potential for further investigation of the Ad5-nCoV vaccine for the control of the COVID-19 outbreak.

To get more info on an article, enter its number. To see more results, enter 'More'. To make another search, enter 'Menu'. To go to
previous page, enter 'Previous'.
```

Figure 2: Example of query processing

3 Framework elements presentation

Sars-engine is based on a 3-level request processing model and includes the use of Facebook Infersent AI[2] for semantic similarity analysis [Conneau et al, 2017]. Infersent operates in two distinct contexts for steps 2 and 3. It is used to get the more similar question in the *step 2* and to find the more relevant sentences of the *search engine corpus* during *step 3*.

3.1 Semiautomatic data extraction

In this section we will discuss how to extract information directly from texts to **fill the table** for *step 1*. In order to give an extract of our work, we have chosen some texts from the World Health Organization website [3] about coronavirus, that can be found in the Appendix. There is a great variety of content: prosaic texts, summaries or schematic texts, or clear indications. This implies that an automatic information extraction process would have to take into account many different styles, in addition to the complexity when adding semantics to the analyzed sentences.

The extraction of information will be manual, sentence by sentence, and the matching process with flexible queries given by the user will be automatic, that is why the *step 1* of the search is **semiautomatic**, as it requires first a manual treatment to fill the table. For each sentence

[2]Infersent repository on Github:
https://github.com/facebookresearch/InferSent
[3]https://www.who.int

Table 1: Example of information in the table. First column is the name of the dimension. Second column (Values) includes the list of values in the dimension. The last column is the meaning of the dimension.

Dimension	**Values**	**Description**
how spreads	[“through droplets of saliva”, “discharge from the nose when an infected person coughs or sneezes”]	How the COVID-19 spreads and produces contagions
prevention tips	[“coughing into a flexed elbow”]	Prevention tips for COVID-19

we will add (if information is relevant enough) an entry in our information database. Each insertion will have the shape of a tuple (**dimension:value:description**), where each dimension corresponds to the row in which the insertion is done, the value is the new value to append to the list of values for that dimension, and the description is an optional parameter to update the meaning of that dimension. Let’s illustrate it with an example. Given the next paragraph:

The COVID-19 virus spreads primarily through droplets of saliva or discharge from the nose when an infected person coughs or sneezes, so it’s important that you also practice respiratory etiquette (for example, by coughing into a flexed elbow).

We are extracting the following information:

- How covid spreads:
  - Through droplets of saliva.
  - Discharge from the nose when an infected person coughs or sneezes.
- Prevention tips:
  - Coughing into a flexed elbow.

As we are doing this process manually, it is easy to distinguish this. However, it is not as easy to process the information about how covid spreads, as the coughing into a flexed elbow advice, because of the way the second sentence is written, the second piece of information would be harder to get.

With this information, it is straightforward to add the following in our table (see Table 1).

3.2 Queries treatment

As our application belongs to a specific domain, the queries made by the user will come (the most of them), motivated by the information extracted. Given the entries added in the previous example from Table 1, they way of interacting with the user should be making queries like:

- Q: How does the covid spread?
  A: Through droplets of saliva, discharge from the nose when infected person coughs or sneezes.
- Q: Does coronavirus spread by saliva droplets?
  A: Yes.
- Q: Any advice to prevent from COVID-19?
  A: Coughing into a flexed elbow.
- Q: Does coughing at the elbow prevents COVID-19?
  A: Yes.

Our goal is to allow the user making variable queries, do not ask questions exactly the same as some that we have predefined. That is, allowing linguistic variability in the questions. We are going to follow a model which focus in some **important words** to know the dimension we are asking for, and depending on the case, which values need to be obtained (for Yes/No queries).

3.2.1 Equivalence class model

We are following an **equivalence class model** inspired by the homonym mathematical concept. The space of queries is a set of strings, and we want to know the answer of every query or how to get the information for that query. Let's make a partition of that space, so that questions with the same meaning are in the same class, that is to say:

$$QueriesSpace = Q_1 \cup Q_2 \cup ... \cup Q_n$$

Where $Q_i \cap Q_j = \emptyset$ if $i \neq j$, and if $Query$, $Query' \in Q_i$, then $Query$ and $Query'$ have the same meaning (and consequently, the same answer). Our algorithm will:

1. Parse the sentence.
2. Look for keywords in its content.
3. Once the keywords are found, the matching with some dimension and specific value or values is done.

3.2.2 Linguistic variability

As the first search (*step 1*) is manual, We can return smarter answers in some cases. We are storing information from bad body temperatures (temperatures such that you likely have COVID), so, if we have a couple of them, it would be more interesting to return the minimum of those temperatures in case of a query as the following:

- Q: Which are fever temperatures to have COVID-19?

In this case we will find the “fever” word, so we will make a projection of the “Bad temperatures” dimension, which has a list of bad temperatures. If this list is [“38”, “38.2”], the system returns the minimum value of the list, 38.

Remark: the system will only find the “fever” word and figure out the dimension from that. This allows the user to make flexible queries, as any other query containing the word “fever” will return the same value:

1. Q: Do I have fever?
   A: 38.
2. Fever temperatures?
   A: 38.
3. Fever and covid?
   A: 38.
4. 1234 fever aaaa?
   A: 38.

The first 3 sentences have a good answer, however, the last question has no sense and has an answer, although the right treatment would be to say that the query has not been recognized. We consider this algorithm based on finding some words good enough for our application, as we think that **linguistic variability is more valuable** than wrong questions are not well treated. Furthermore, a bad-formed question is something easy to detect and correct by the user, and it is not critical at all this behavior.

- Our system **is not sensitive to capital letters**, so more flexible inputs are available. However, it is recommended to avoid punctuation marks.
- About **preferences in case of ambiguity** in dimensions, the semiautomatic parse of queries will find first more specific things, for example, given some query as “*Tell me ventilation tips*”, the system will match with some prevention tips as the keyword tips appear in the sentence. Moreover, it will also match with the ventilation tips dimension, which is an specific case of prevention tip, and will be the displayed answer.

3.2.3 Matching with queries

Although each query is treated separately, we are going to illustrate the most common case of search in *step 1*:

Table 2: Good places dimension in the database.

Dimension	**Values**	**Description**
Good places	[“open places”, “parks”]	Good places to go when a pandemic

Given a query, some keywords in it point out the dimension in which we have to search.

- Q: Which are **good places**?

The words “good” and “places” point out the dimension: “Good places”, detailed in Table 2. As no more information is given, the expected behavior is answering one good places. By default, the first value from **Values** column is displayed, so “parks” is the answer. There are much more ways to look for information in the dimension, the three most common cases are:

- If the word “all” appears in the query content too, all the values of the dimension will be displayed. Example:
  Q: Which are all the good places to go?
  A: Parks, countryside, open places.
- If not, if some of the values also appears in the query, it will be considered as a Yes/No question. Example:
  Q: Are parks good places to go?
  A: Yes.
- Else, some of the values from the dimension will be displayed. Example:
  Q: Some good place to go?
  A: Parks.

Although this algorithm tries to be homogeneous for all the dimensions, some of the displaying options actually are not reallistic: given the dimension **prevention advice**, the user is not going to ask if some concrete tip is an advice or not. The user would like to read some complete tips, that is to say, there will not be Yes/No queries in this dimension. In any case, it would be difficult for the user to formulate a right Yes/No query, as he would have to ask something like:

- Q: Is protect yourself and others from infection by washing your hands or using an alcohol based rub frequently a good prevention advice?
  A: Yes.

3.3 Infersent model : cosine similarity

The Infersent model encodes our words as vector and our sentences as vectors composed of 4096 features. To find similar sentences from a given one, we are going to use the cosine similarity, a popular technique in Vector Space Model [Singhal, 2001] based on the comparison of the angle between two vectors. In an Euclidean space, we can perform the dot product between two vectors $A \cdot B$, which is the sum of the coordinates of their element-wise product.

$$(1, 2, -1) \cdot (1, 0, 1) = 1 \cdot 1 + 2 \cdot 0 + (-1) \cdot 1 = 0$$

This product is equal to the following formula:

$$A \cdot B = ||A||\ ||B||\, cos\, \theta$$

Where $||A||$ is the norm of A and θ the positive angle between the two vectors. The cosine similarity between two vectors is:

$$similarity = cos\, \theta = \frac{A \cdot B}{||A||\ ||B||} \in [0, 1]$$

3.4 Build a Search engine corpus

3.4.1 Leading principles

Here are presented the main steps we followed to build our search engine corpus. As mentioned before, the corpus contains pre-processed coronavirus related sentences. Those sentences can be extracted from a corpus of coronavirus related texts (scientific or public press articles for example).

The chosen method is to select articles in function of a pre-established glossary that is made in function of the future search engine use. The glossary can contains any word that would be likely to be searched in the context of a specific application. For example, it could only contain scientific related terms if the search engine is made to help searchers to find out similar works that have been made in their specific topic.

Then, it is assumed that the more an article contains words of this pre-established glossary, the more it is relevant for our application. Therefore, documents are sorted by the relevant words (words of the glossary) number of occurrences. Over a basic corpus that contains a lot of documents that are not all necessary related to our problematic, we can only keep a few of them in order to reduce next steps computation cost. At the end, from those selected articles we will keep their bodies, abstracts, titles, authors and fill the blank values with "unknown" likely labels. We associate an article number to each document. Then, we tokenize the documents into sentences and filter them to keep only the ones that contain relevant words. In any case, the full articles are still stored a dataframe saved as a file. The sentences are stored in the form of a dictionary that associate to each sentence its article's number. This dictionary is also saved as a dataframe.

Finally, an Infersent model is called to build a dictionary of our sentences vocabulary, and from it it encodes each of the sentence as a 4096 features vector. The encoded sentences are stored in a heavy file.

3.4.2 Application's *search engine corpus*

Our initial corpus was built by sampling documents in the CORD-19 dataset (`https://www.kaggle.com/allen-institute-for-ai/CORD-19-research-challenge`). It was made by The White House and a coalition of leading US research groups. It's being updated oftentimes and covers all type of documents related to SARS-CoV-2. As it contains some documents written in German, French, *etc.* the glossary filtering allows to keep only with the english ones. A task list was deployed by the US governmment to guide the challenge. This task list is mainly composed of SARS-CoV-2 questions whose answers could help for scientific research (example : *What do we know about COVID-19 risk factors?*). Actually, our application could partially be seen as integrated to this data science challenge.

The more than 7 million of documents (when this report is written) are given in a *json* format. It allows us to extract principal metadata (authors, abstract, *etc.*) easily. For technical reasons, we could not handle more than 750 documents, that were selected from a 10 000 documents sample (according to their relevance). The final corpus is composed of 155635 sentences.

Our glossary contains a large list of symptoms related to the disease, taken from a CORD-19 related notebook (`www.kaggle.com/justmax1604/covid-text-extraction`), which was complemented by other covid related terms presented in the *corpus.txt* files.

4 Make it work

You have two options to make use of our framework : use our *search engine corpus* or build your own thanks to the *build_corpus.ipynb* file. If you want to build your own corps by using our method, you can fine the *json* CORD-19 documents that you can download online. You need to take the *pdf_json* folder and to place it in your working repository.

For both corpus building and application python codes, the main argument is the path of this repository. For the one to build a corpus, you also need to specify the number of documents to sample, and the final number of document so select. For a personal computer using an Intel i7 7500u CPU, the total code can be run in about 3 hours and 11 minutes for 750 documents. Note that the main limitation when increasing the number of documents is your RAM (with 16

GB, it was impossible to select 2000 documents). Of course, you can modify the code to create a bigger corpus divided into multiple files.

For the two codes you will also need basic Infersent features. The trained model (*infersent-2*), a vector model of english words and the InferSent class can be downloaded online through Github and Facebook AI public files repository.

The *corpus of questions* and the *search engine corpus* are generated by the build_corpus.ipynb code, over different forms (pandas data frame, dictionary and/or infersent encoding).

To run our scripts with success, ensure yourself to have in your repository the necessary files referenced in Table 3. Some of them can be download from our GitHub repository (G.R.) : `https://github.com/iliassarbout-engine`.

Finally, you may need to install a couple of python libraries in your environment, that are all mentioned in the first cell of each our our scripts.

5 Future improvements

5.1 Improve results

- Exploit Natural Language Inference Infersent module to filter the *search engine* results.
- In *step 1*, improve the numeric queries like "do i have COVID-19 if I am 40 degrees?" Now we are displaying a single number (as "38"), instead of "Yes". We should parse if there are numbers in the sentence and compare if they are in a range of values. According to the percentile in which it is (respect of the other values from the list), we could display a probability or a more realistic output as "You probably have COVID-19" or "You probably do not have it".
- Integrate a method to avoid step 2 during query processing if the found question is not that similar to the one formulated by the user. It could be by using a threshold on the cosine similarity value or by checking if the most important words (that could be extracted with *nltk*) of the user query are contained in the question
- For a given sentence found during step 3, we can assume that its corresponding articles contains other sentences related to the answer. Instead of printing only this sentence, we could print the 2 or 3 most similar sentences, separated by a '[...]' contained in the articles (and use again a threshold).
- During step 3, instead of looking for sentences similar to the user query, it may improve the results to transform it in a positive statement, and also to find synonyms of this positive statement (maybe all of this could be implemented using *nltk*).

- We could add weights to the *corpus of questions* elements to give values to frequently asked questions and also propose more than a single question to the user.
- Considering building Infersent vocabulary and embeddings is quite long, the vocabulary used for question similarity analysis (step 2) is the same as the one used for *search engine corpus*. Actually, it is quite probable that the set of words used in the *corpus of questions* include any that could be used for a query. However, build a proper dictionary containing only the corpus of questions words would probably improve the question similarity analysis (step 2). The reason why it was not done is the time needed to encode vocabulary and sentences, that does not allow to repeat this process every time we add a question. For a web based application, we could imagine stocking the new questions/answers in a batch-sized temporary dictionary and update Infersent model at regular intervals.
- Analyze how the *tokenize* argument of the *Infersent*'s embedding function impact on the quality of our search engine. We did not have enough computation resources to do it.
- It was explained that we sorted documents and keep only the most relevant ones. However, to cover more information, we could first tokenize the documents into sentences and take the n more relevant ones, instead of eliminating directly entire documents. Moreover, it would allow to have a better control on the dataset size, while document's size have an important variability.

5.2 General structure and optimization

- Filter documents licences to keep only with free redistribution articles.
- Add an option to make directly researches over the *search engine corpus*.
- Define tables to interpret better user's choices in case he make mistakes ("Yess" instead of "yes" for example).
- Improve full articles reading (it would be time-consuming to implement it for python, since on a web-based application it would be easier).
- During *step 3*, instead we compute similarity of user's query with the whole search engine corpus before sorting. This process was quick since our final corpus contains 155635 sentences extracted from 750 articles, but actually has the biggest algorithmic cost among the operations that take place during the user application. When using the whole CORD-19 dataset composed of more than 6 million documents, this sorting will be quite long so we should take care of extracting directly the most similar sentences when computing the similarities. If we only extract the 100 most similar sentences, we could do it in a linear complexity ($O(n)$) instead of a quadratic one.

- As the *pickle* library could not support very heavy files, the code use both *pickle* and *joblib* elements. We will soon make use only of *joblib*.
- Create a separate class containing only the Infersent model, in order to avoid the initialization of a proper Infersent model for *search engine corpus* elaboration. It would be element of both I*nferSent_corpus* and *Corpus_questions* classes.
- There are a few residues due to work in progress in the code (use of Pool library for multiprocessing or about lemmatization of the sentences) that could be deleted.
- Delete *sentence_no* column of sentences dataframe as the index gives the same information.

References

[Conneau et al, 2017] A. Conneau, D. Kiela, H. Schwenk, L. Barrault et al (2017). *Supervised Learning of Universal Sentence Representations from Natural Language Inference Data.* Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing.

[Singhal, 2001] Singhal, Amit (2001). *Modern Information Retrieval: A Brief Overview.* Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4): 35–43.

[Wang et al, 2020] Wang, L.L., Lo, K., Chandrasekhar, Y. et al (2020). *CORD-19: The Covid-19 Open Research Dataset.* ArXiv.

Appendix

In this appendix we include the texts from which we have extracted the information to fill manually the table structure from Section 3.1 in *step 1*. This information comes from the World Health Organization website. We also include the Table 3 that summarize the framework files information.

Coronavirus disease (COVID-19) is an infectious disease caused by a newly discovered coronavirus.

Most people infected with the COVID-19 virus will experience mild to moderate respiratory illness and recover without requiring special treatment. Older people, and those with underlying medical problems like cardiovascular disease, diabetes, chronic respiratory disease, and cancer are more likely to develop serious illness.

The best way to prevent and slow down transmission is to be well informed about the COVID-19 virus, the disease it causes and how it spreads. Protect yourself and others from infection by washing your hands or using an alcohol based rub frequently and not touching your face.

The COVID-19 virus spreads primarily through droplets of saliva or discharge from the nose when an infected person coughs or sneezes, so it's important that you also practice respiratory etiquette (for example, by coughing into a flexed elbow).

Prevention.

To prevent infection and to slow transmission of COVID-19, do the following:W ash your hands regularly with soap and water, or clean them with alcohol-based hand rub. Maintain at least 1 metre distance between you and people coughing or sneezing. Avoid touching your face. Cover your mouth and nose when coughing or sneezing.S tay home if you feel unwell.

Symptoms.

COVID-19 affects different people in different ways. Most infected people will develop mild to moderate illness and recover without hospitalization. Most common symptoms: fever, dry cough, tiredness. Less common symptoms: aches and pains, sore throat, diarrhoea, conjunctivitis, headache, loss of taste or smell. Serious symptoms: difficulty breathing or shortness of breath, chest pain or pressure, loss of speech or movement. On average it takes 5–6 days from when someone is infected with the virus for symptoms to show, however it can take up to 14 days.

Table 3: Necessary files description

File	**Short description**	**How to obtain it**
build_corpus.ipynb	Build a search engine corpus and initialize a corpus of question	Our G.R.
main.ipynb	Launch the application	Our G.R.
infersent.pkl	Infersent pre-trained model	Get it from Facebook AI public repository
crawl-300d-2M.vec	Vector representations of english words	Get it from Facebook AI public repository
df_questions.joblib	Pandas dataframe containing corpus of questions with associated answers	Our G.R. or generate it with build_corpus.ipynb
documents.pickle	Pandas dataframe containing the documents (names,number,authors,abstract,body, number of relevant words)	Our G.R. or generate it with build_corpus.ipynb
document_sents.pickle	Pandas dataframe containing search engine corpus (sentences with their associated article number)	Our G.R. or generate it with build_corpus.ipynb
glossary.txt	Is added to the covid symptoms list for documents selection during the construction of a search engine corpus	Our G.R./use a custom one
embeddings_questions.joblib	Corpus of questions encoded with Infersent vectors	our G.R. or generate it with build_corpus.ipynb
embeddings.joblib	Search engine corpus encoded with Infersent vectors	Download link in our G.R. or generate it with build_corpus.ipynb
models.py	Contains the infersent class	Infersent github repository or our G.R.