"When will the pandemic end?"

An italian chatbot about Coronavirus

Ludovica Binetti
l.binetti@studenti.unipi.it

Stefano Moscatelli
s.moscatelli2@studenti.unipi.it

Lorenzo Di Gianvittorio
l.digianvittorio@studenti.unipi.it

Digital Humanities, Department of Philology, Literature and Linguistics

February 6, 2021

Abstract

In this paper, we present a chatbot which is able to answer to Covid-related questions by matching the embedding of a query with the ones of the possible answers taken from a corpus. By implementing different types of models, our goal is to discover the best one and suggest further improvements and possible useful application of it.

1 Introduction

In its most straightforward definition, a chatbot is a simple AI implementation that is able to analyze a textual (or oral) input and generate an appropriate response. In our case, the chatbot receives and returns just textual inputs/outputs. What led us develop a chatbot is the versatility of this tool: once implemented, it can be applied in many different fields, depending on the type of data it is fed with. Besides, we decided to implement a chatbot about Coronavirus because we believed it could be of extreme help in the current times, where people always have doubts and perplexities about COVID-19. Our goal is to provide a tool that would let citizens find, in an easy and quick way, reliable answers to their questions and, at the same time, release private clinics and hospitals call centers from the burden of receiving many calls during the day. Our chatbot isn't, of course, a general-purpose chatbot (as it would definitely be too difficult to implement) but it focuses on giving consistent answers to Covid-related issues. It can be considered as a database of sentences which are

extracted and returned according to their similarities with the user-generated question. As it will be better clarified in paragraph 3, the "question-answering bot" does not generate an answer on its own but it returns 2 different types of outputs: either an answer taken from a most frequent asked question, or a sentence extracted by an article. In a text analytics perspective, the main goal is to achieve question answering by giving to the sentences (contained in the training corpus) a good and satisfactory embedding representation.

2 Data collection

The first step for building such a system was to collect textual data. Since the chatbot deals with a crucial problem of our era, we identified (in a complete arbitrary way and according to what seemed more accessible) two main reliable and official sources of information:

- the Ministero della salute website, from where we crawled and downloaded both news and FAQs;
- the Epicentro ISS website, from where we downloaded articles;

The decision of mixing up FAQs and articles was mainly because we wanted to work on different types of sources. However, in order to download texts from the earlier mentioned resources, we initially went through the source code in order to identify the exact location of the text we had to extract. We then used the urllib package [1] and the Beautiful Soup library [2] to open URLs and pull out text from the HTML source respectively. The first challenge was to understand the structure of the HTML source and to identify whereabout the pieces of text we wanted to extract were located. Another frequent problem we encountered when downloading articles was the HTTP 403 forbidden error, which we solved by applying a delay between the requests of 0.5, 1 or 5 seconds to download the content of a certain page, which it is, among other things, a requested netiquette procedure. The extraction of FAQ was, instead, quite smooth and definitely easier than articles download: once we identified the tags that incorporated the questions and the corresponding answers, we simply accessed to all the webpages of Ministero della Salute website, that contained FAQs about many different aspects of Coronavirus. The extracted FAQs were saved in a dictionary whose keys were the frequently asked questions and whose values were tuples: the first element of the tuple was the answer, whereas the second one was the link of the webpage from where both question and answer were extracted. We actually didn't use the links for anything in particular, but we thought it might have been worth keeping them in case of further implementations: for example, the chatbot might exploit the links by retrieving them and provide them to the user in case he/she wanted to examine the actual source where the info was taken from. Once the data collection was completed, we obtained:

- 100 articles (that were published between the 11th of July and 14th of January) and 154 FAQs (updated to 26th of January) from Ministero

della salute website;

- 65 articles from Epicentro - Istituto Superiore della Sanità one.

The articles were all unified in a single .txt document that contained a list of sentences, unlike the FAQs that were stored in a dictionary as described above. Unfortunately, to avoid to waste time in manual preprocessing of our data, we decided not to clean the articles from "noisy sentences". For this reason, it could happen that the chatbot answers with half-way sense answers which are, in some ways, typical of digital newspapers articles: for example, doc2vec model returns answers like "Aggiornamento al 18 dicembre 2020, (pdf 1,6 Mb)", "Lista interventi", "Queste in sintesi le novità introdotte dal nuovo DPCM.", "Guarda i video" and so on.

3 Pre-processing

The pre-processing of our training corpus was run locally for every implemented model (apart from neural networks which simply needed raw text), so we won't explain any further here. We just wanted to specify that we didn't POS-tagged our corpus in order to work with the lowest n-dimensional space as possible. Also, we didn't lemmatize our data because even if we did implement a stemmer (we tried with Snowball from nltk), we noticed that there wasn't much difference in the output. An important thing to keep in mind is that, for every algorithm, we created a list containing all the sentences from the articles and only the questions from the FAQs. In this way, we could compute a similarity between the user input and a sentence that the model would have returned as an answer, except from the questions: if the user input was similar to one of these, the answer for that specific FAQ was retrieved from the dictionary and returned as an output. In fact, it's obvious that if a user question is similar to one of the FAQs, the chatbot already has a good and well-articulated answer to return.

4 Model building

In this section, we are going to briefly describe the models we implemented to generate embeddings for both our sentence corpus and the user-generated input. The similarity score was in general computed with cosine or built-in model functions that exploit cosine similarity (like doc2vec most_similar).

4.1 Count Vectorizer (by sklearn)

Count Vectorizer [3] is a model that produces sparse document embeddings by looking at the word frequencies and that creates a term-document matrix (where the document is either a sentence from articles or a question from FAQs). We didn't explicitly preprocess our data, but used, instead, the default parameters of Count Vectorizer function that automatically preprocess the input text. In

particular, we would like to stress the fact that we used n-grams (from unigrams to trigrams) in order to capture contextual information of a word: Count Vectorizer, in fact, usually uses a bag of words representation that wouldn't allow to keep track of words order. We also set a min_df=2 to remove hapax and a max_df=1000 to remove stop words (of course, we previously checked how many times, on average, typical stop words like "la", "del" or "in" appeared in different documents).

4.2 Tf-idf (by sklearn)

We decided to use the scikit library beacuse we wanted to explore as many different libraries as possible. When implementing LSI model, we also partially used tf-idf model from the Gensim library: this is why we didn't want to use it again. Similar to what we did for Count Vectorizer, we performed a basic preprocessing by setting up parameters in the TfidfVectorizer function [4]: min_df = 2 (to remove hapaxes) and a max_df = 1000 (to remove stop words). However, by setting up this last parameter we were just being extremely precise: even if we hadn't set it up, the final output would have been the same (or anyway really similar). In fact, tf-idf tends to spontaneously assign a lower score to words that are really frequent in many documents (which are, for the record, stop words).

4.3 LSI (aka LSA - by Gensim)

Here, in order to clean our text we used a built-in gensim function (simple_preprocess) that tokenizes the text and brings everything in lowercase. For this particular model, we also had to:

- Build a Dictionary;
- Transform the text into a Bag Of Words representation;
- Weight the vectors with Tf-idf values.

After applying the LSI model [5] to our tf-idf vectors, we built an indexed matrix similarity in order to facilitate the retrieval of the chatbot answer according to the user-generated query.

4.4 Doc2Vec (by Gensim)

Similarly to LSI, for doc2vec [6] we used the simple_preprocess function. Even if doc2vec is a neural network, we had to preprocess our corpus because we needed to feed doc2vec algorithm with a special representation of the text created by TaggedDocument function which associates to every training document/sentence an id and which must take as input a list of token. In the model building phase, the context window of doc2vec was set to 5 and the vector_size to 100. In the training phase, after running different experiments in which we

increased the number of epochs (iterations), we finally set it up to 70 as we didn't notice any particular improvements above this value.

4.5 BERT (by Google)

BERT is a powerful architecture released in 2018 with high performances on tasks related to Natural Language Understanding. It consists of a stack of encoders that exploit bidirectional self-attention. In this experiment, we have chosen it for its ability to create contextualized word embeddings. We didn't actually used the original BERT model, but 2 different and more recent implementations of it:

- RoBERTa (Robustly optimized BERT approach)[7][8]: it is a model based on the same architecture of BERT but pre-trained in a slightly different way. Among many different RoBERTa implementations, we chose one optimized for Information retrieval (as it is used in Bing search engine). Given a query (which can be a single keyword, a sentence or a question), it ranks, in a decreasing order, the most semantically similar documents.
- BERT XXL italian [9]: a model trained on over 2 billion italian tokens.

4.6 Universal Sentence Encoder Multilingual (by Google)

This is a model that improves the standard Universal Sentence Encoder of 2018 with a massive training on 16 different languages [10]. This model suits many different tasks including translation, since the training provides the same embedding space to different languages. Here we focus on its ability on semantic retrieval tasks. The preprocessing of the model includes various level of tokenization and normalization through the SentencePiece tokenizer.

5 Building the test set

As we collected the data from the web and we therefore knew the general topic of the texts we downloaded, there was the risk that we would have built a "compromised" test set: for example, by putting too many questions that were present (or viceversa absent) in our sentence corpus. This is why we decided to ask to our relatives and friends to provide us some of their own questions. In this way, we tried to build a test set as "neutral" and representative as possible: in fact, another advantage of this solution was, of course, to simulate and test our chatbot on "real" user-generated questions, which are doubts that people would have in real life about the recent pandemic situation. In doing this, we obtained a test set composed of 42 questions, that we iteratively asked to every model we implemented.

6 Test phase

In order to evaluate the performance of each of them, we decided that each of us would have assigned independently a 1 if the answer was correct, a 0 if it was wrong and a "?" if we were in doubt as it could have been both, either wrong or correct. In case the three of us didn't agree on the answer given, the simple majority of the votes would have won (e.g. if we had 0-1-1, the final evaluation would have been 1, therefore correct). Of course, we have to be aware of the fact that when the three of us agreed on a bad answer returned by the chatbot we didn't know whether it was due to an inappropriate vector representation (that was not able to capture the real meaning of the sentence) or due to the fact that an actual answer didn't exist in the corpus of possible ones. We can see in Table 1 that there are actually some questions with no correct answer in none of the models. This made us think that, probably, the answer didn't genuinely exist: this is why we roughly checked by running a Find search in our .txt files of the most relevant words contained in the user question. Afterwards, in order to evaluate the final performance of every model, we decided not to take into consideration the questions we didn't find a proper answer for, which were exactly 12. In the end, we could state that Tf-idf is the best model (with

QUESTIONS
Quanto tempo resta nel corpo il coronavirus?
Quando finirà la pandemia?
Chi si è vaccinato può trasmettere la malattia?

Table 1: Some of the questions none of the models found an answer for. (Respectively: "How long does the coronavirus survive in our body?", "When will the pandemic end?", "Can vaccinated people transmit covid-19?").

43% of accuracy), followed by Count Vectorizer and BERT XXL (see Table 2). The worst ones are indeed Doc2Vec and Universal Sentence Encoder which obtained an accuracy of 0%. In general, we had the impression that by using both articles and FAQs the performance of the models decreased rather than using just FAQs. It might probably due to the fact that, as we already stated in

	Accuracy
CountVec	0.33
Tf-idf	0.43
LSI	0.27
doc2vec	0.00
BERT XXL it	0.33
RoBERTa	0.27
USEM	0.00

Table 2: Accuracy of the models.

section 2, the text of articles is full of noise: there were cases in which, instead of retrieving and returning an existing related FAQ, it would just give a no-sense sentence taken from the articles corpus.
Also, we were quite surprised that doc2vec performance was so bad and we wanted to try to feed it with the only FAQs, for the reasons discussed above. Unfortunately, its performance wouldn't improve, so we had to suppose that the model needs to be fed with more training data.

7 Conclusions and what comes next

After selecting the best model, we decided to provide the chatbot a nicer interface using tkinter package. What we would be interested to understand now is whether the chatbot we tried to implement would be a useful resource for the Italian government to face the current sanitary emergency. We strongly believe that if the model would be fed with only (and probably more) FAQs not only it could improve its performances, but it could represent a valid tool to quickly look for reliable information: in fact, potential users, instead of crawling the Ministero della salute website (or even other reliable and official sources such as Governo italiano website or ISS faq section) to find the appropriate answer to their doubts, could save lots of time if they had an automatic system which would automatically crawl all the websites for them and give an (almost) immediate answer. Moreover, it could help the government to collect data about people's questions and, therefore, update the websites with relevant information for the users.
With these goals in mind, we finally connected our chatbot to a well-known instant messaging platform: Telegram. By making it available in such a popular app, we thought that people would be more motivated to use instead of merely providing it on a website.

References

[1] "Urllib.request documentation," https://docs.python.org/3/library/urllib.request.html#module-urllib.request.

[2] "Beautiful soup documentation," https://www.crummy.com/software/BeautifulSoup/bs4/doc/#navigating-the-tree.

[3] "Count vectorizer documentation," https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.

[4] "Tf-idf sklearn documentation," https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

[5] "Lsi documentation," https://radimrehurek.com/gensim/models/lsimodel.html#gensim.models.lsimodel.LsiModel.

[6] "Doc2vec documentation," https://radimrehurek.com/gensim/models/doc2vec.html.

[7] "Roberta model," https://huggingface.co/transformers/model_doc/roberta.html.

[8] "Roberta pretrained model," https://www.sbert.net/docs/pretrained_models.html.

[9] "Bert xxl model," https://huggingface.co/dbmdz/bert-base-italian-cased.

[10] "Usem model," https://developers-it.googleblog.com/2019/07/universal-sentence-encoder-multilingue-il-recupero-semantico.html.