Artificial Intelligence II
Assignment 4 Report

Andreas - Theologos Spanopoulos (sdi1700146@di.uoa.gr)

January 17, 2021

Exercises 1 & 2

For these exercises, publically available pre-trained models from the python library Sentence Transformer were used. A list with all the available models and their respective statistics, can be found here.

Preprocessing

From the dataset, the following fields are parsed and stored in the class "CovidArticle":

- ID
- Title
- Abstract Text, tokenized into sentences
- Body Text (section names and paragraphs), tokenized into sentences

For every piece of text, the following preprocessing pipeline is followed:

1. Remove URLs
2. Remove references to bibliography
3. Remove multiple full stops (e.g. ...)
4. Remove the "et al." string
5. Remove figure references

Numbers 1, 2 and 5 were removed because they offer no use in retrieving a sentence containing them. Numbers 3 and 4 were removed because they break sentence tokenization.

Every piece of text is broken down into sentences. A list containing all the sentences can be accessed in the CovidArticle.text getter method. This format is convinient because it allows the sentenced to be fed directly to the sentence transformers.

Sentence Embedding Approaches

The 2 models that were finally chosen, are:

1. stsb-distilbert-base. This model was chosen because of its good performance, and its high speed in computing the embeddings of sentences (4000 senteces/sec on V100 GPU).

2. stsb-roberta-base. This model was chosen as its the second best model regarsing the STSb performance, and its speed is acceptable (2300 sentences/sec on V100 GPU).

Both models are Sentence-BERT models, which basically means that they use BERT-like pre-trained models to compute the sentence embeddings.

Now let's discuss about how the models are used to retrieve relevant text for every query. The pipeline used is as follows:

1. Filter our articles that are irrelevant to the subject query.

2. From the articles kept, find the best one and return it.

Step 1

For every article, a "summary" is computed. This summary is the simplest possible: it's a list consisting of sentences which are:

1. The title of the article

2. The tokenized sentences of the abstract of the article

3. The sections of the article, each being treated as a standalone sentence.

Then for every model, we compute the sentence embeddings for each sentence in the summary. These embeddings are used for filtering our irrelevant articles: Articles where the highest cosine similarity with any sentence of the summary is lower than a specified threshold, are discarded. This helps us keep only relevant articles and therefore compute less sentence embeddings, as it is quite a costly operation, especially on low-end hardware.

Step 2

For the articles that have "survived", compute the sentence embeddings for the whole text, and pick the one that has the highest cosine similarity with the query, on any sentence. Then, consider that sentence the passage. After that, start compuring the cosine similarity of adjacent sentences with the initial passage, and if they are above a 0.5 threshold, append them to the passage. This process repeats until either the threshold of 0.5 is not met on both sides, or we run out of the section in which the inital passage belonged.

Comparison

Some test queries have been hand-written for evaluation purposes. Those queries and the corresponding articles in which their answers lie, are listed in the queries.txt file.

In the task of finding the correct article containing the answer to a given query, the models performed:

Query \ Model	stsb-distilbert-base	stsb-roberta-base
Query 1	✓	✓
Query 2	✓	✓
Query 3	✓	✓
Query 4	✓	✓
Query 5	✓	✓
Query 6	✓	✓
Query 7	✗	✓
Query 8	✓	✗
Query 9	✗	✗
Query 10	✓	✓
Total Accuracy %	80%	80%

Let's also take a look at the results in the task of returning also the correct passage along with the correct article:

Query \ Model	stsb-distilbert-base	stsb-roberta-base
Query 1	✓	✓
Query 2	✓	✓
Query 3	✓	✓
Query 4	✓	✓
Query 5	✓	✓
Query 6	✓	✓
Query 7	✗	✓
Query 8	✓	✗
Query 9	✗	✗
Query 10	✓	✓
Total Accuracy %	80%	80%

The time required to compute the embeddings of all the summaries is

- 15 minutes for stsb-distilbert-base
- 27 minutes for stsb-roberta-base

The average time required to return the result of a query for every model is

- 17.75s for stsb-distilbert-base
- 24.91s for stsb-roberta-base

Note that these times get faster as more queries come in, because article text embeddings get saved in a dictionary.

Error Analysis

For an uncorrect article/passage to be returned, there are 2 possible errors that might have occurred:

- In Step 1, the correct article got filtered out.
- In Step 2, the correct article was kept but did not contain the sentence with the maximum cosine similarity.

Let's take a look in which case each of our query failures belong to.

By un-commenting the 2 print statements in the "find_best_article()" function, we can see which articles were found relevant. The others were filtered out. Let's make a table for every model to see what is going on:

stsb-distilbert-base

Failure Reason / Query Failures	Step 1 Error	Step 2 Error
Query 7		✓
Query 9	✓	
Total	1/2	1/2

stsb-roberta-base

Failure Reason / Query Failures	Step 1 Error	Step 2 Error
Query 8	✓	
Query 9	✓	
Total	2/2	0/2

From this small sample, we can deduce that the "summary" tecnhique does not work perfectly. This is expected, as the title + the abstract + the section titles do not capture the entirety of an article. For example, we may have an article about coronaviruses, but in the introduction there is a reference to the Spanish Flu, and the amount of casualties it caused. Since this information is irrelevant to the article itself and the pieces that make up the summary, it will be filtered out. Of course, there are way better methods to create

summaries, like summarizers, etc. These haven't been tried, as I am running low on time. One solution could be to decrease the similarity threshold, so that more articles get included. This will slow down significantly the running time, so it is not an option. Therefore, improving the summary of each article should be the way to go for improving this text retrieval mechanism.

One other thing that could be done is to get rid of the summaries, and just pre-compute once the sentence embeddings for every article. On the Colab GPU, this procedure takes about 2 hours for a model like *stsb-roberta-base*. Since cosine similarity is not a very expensive operation, this would lead to the best results, as all the articles would be taken into account, but slower (on average) as it would compute $O(n)$ cosine similarities for every query, where n is the number of article.

The above idea would technically give the most accurate results. Will it? There is a small catch. The problem lies on the selection of the "best" sentence. The sentence with the highest cosine similarity with the query is not always the answer to it. This problem becomes apparent when the embeddings have not been trained on specific domain-knowledge data, for example, text containing biological terms. The same problem would occur in the above models, had more queries been created. Thus, apart from the issue of identifying quickly the relevant articles, one needs to make sure that the remaining articles can be fully understood, that is, the sentence embeddings will capture all of its semantic meaning. This can be achieved by fine-tuning the models on the specific domain-knowledge data.

Conclusion

Taking into account the above results, model "stsb-roberta-base" can be considered as the best out of the two models, since

1. It filters out the most sentences (error analysis showed that it keeps around 5-25 sentences per article).

2. Even though slower, it finds the *exact* answer for most queries, unlike "stsb-distilbert-base" which sometimes finds *approximate* answers.

3. Once most sentence embeddings will have been computed, it will become faster than "stsb-distilbert-base", since it will only have to find the cosine similarities with many less articles.

Exercise 3