Document Ranking Using a Neural Retrieval Model

Jenna Ingels Robbe Lauwers

December 16, 2021

Abstract

As technology develops and the amount of data hosted increases, the demand for precise search applications increases. This paper describes our process in implementing search ranking for use on top of Lucene using the Sentence BERT framework. We used the Sentence Transformers library for Python in our implementation. The reason for this choice of topic is that, while Neural Networks are a recent development in the field of information retrieval, they are quickly becoming a powerful all purpose tool. It is therefore of utmost importance for us as developers to keep up with developments in this field. Our goal is to improve upon the performance of the Lucene search library, primarily where synonyms and polysemes are concerned.

1 Introduction

"Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.", this is an extreme example of lexical ambiguity. Naturally evolved languages have no shortage of the sort. Then it will come as no surprise to anyone who has worked with computers that the processing of such Natural Languages is a complicated problem in the information industry. After all, humans usually communicate with computers using programming languages. These languages are syntactically simplified when compared to natural languages. This allows us to bypass any trouble arising from such ambiguity entirely.

One such field where avoiding ambiguity entirely is impossible, is the field of information retrieval. The information that has to be retrieved, is written in natural language after all. How is a computer program to know that the bank a customer meant to visit on a sunny day was a river bank. This is where neural networks and other artificial intelligence approaches come in.

Sentence BERT is a 2019 development based on BERT. BERT was first released to the public in 2018 by Devlin et al. at Google.[RG19] BERT, short for Bidirectional Encoder Representations from Transformers, is a machine learning technique for natural language processing developed by Google researchers. It should then come as no surprise that the framework is uniquely suited for our purposes of search refinement.

2 Implementation overview

The Github repository for our solution is at `https://github.com/RobbeLauwers/Information_retrieval_2`. However, most of the code was tested in Google Colab here `https://colab.research.google.com/drive/1c2DH0ytXTdclzNoDIaTNI2d_eU5XyDun?usp=sharing`. For an explanation of which code cells should be used, refer to the readme on github. The presentation is on github as Presentation.pdf.

According to the Sentence transformers documentation[Rei21d], an information retrieval system can be implemented by using a bi-encoder to find relevant documents for a given query. After this, cross-encoders are used to re-rank the results. This is otherwise known as augmented SBERT[Tha+21]. Considering that the dataset in the assignment is already the result of a search, executed with Lucene, we do not need to use bi-encoders and only need to re-rank the results.

While the sentence bert library allows the use of pretrained cross-encoders, these models can also be trained further. We will use a pretrained model as starting point and fine-tune it on the provided dataset.

3 Testing

As mentioned in the overview, we will use a cross-encoder to judge the relevance of queries and documents. A cross-encoder starts with a pre-trained model, but can be trained further to better fit our dataset. We first extract the relevant info from the training data dev_data.csv: we need query/document pairs and the relevance label. We then create a cross-encoder from a pre-trained model and train it further on the provided data. Several models and training parameters have been tested and will be discussed in the next section.

For the tests, we will select hyperparameters based trial and error based on previous results, but also based on the SBERT documentation. For example, take the evaluation function that is used during training: the documentation[Rei21a] has a list of all evaluation functions, but also explains what each function is best used for.

We did the first few tests on our own computers, but it was rather slow, so we decided to use Google Colab. This allows us to run the machine learning on a GPU owned by Google. We were provided with the same GPU during all of our tests, an NVIDIA Tesla K80. The drawback of using Colab is that there are hidden usage limits. Because of this, we often kept our tests as small as possible by only using part of the provided data, to ensure that we could do all test we wanted at a reasonable speed.

3.1 Evaluating our solutions

We wanted implement two methods to analyze how good a solution is: we will check the precision and recall of the binary labels, and we implemented rank biased overlap[Jos21] to check if the re-ranking is good. However, rank based overlap was rather slow for the amount of tests we wanted to do, as it involves sorting all the data, so we eventually decided not to use it.

3.2 Pre-trained models

The cross-encoder we use starts with a pre-trained model that we then finetune. A list of pre-trained models is available in the SBERT documentation[Rei21c]. This list contains models trained on several different datasets, a general overview of what the datasets contain, and several models for each dataset. We initially picked models trained on the MS MARCO dataset, because this was trained for information retrieval. Considering that we will test several hyperparameters and would like to test as many combinations as possible, we initially picked one of the models with the fastest speed: `cross-encoder/ms-marco-TinyBERT-L-2-v2`[Rei21b].

In our first tests, we trained the model on the development data for one iteration using default hyperparameters, then used the model to generate scores for the same data. The generated scores were between 0 and 1, and were rounded to the nearest binary label. The test results were rather poor. While the precision was around 0.65, the recall was well below that, 0.37.

Because the result was rather poor, we decided to use a new pretrained model before testing different hyperparameters. While SBERT offers several more pre-trained models, the only relevant ones were QNLI trained on the SQUAD dataset. This model is trained on data where one sentence is a question, and the other is a Wikipedia paragraph containing the answer. From the two models available, we picked
`cross-encoder/qnli-distilroberta-base`. Surprisingly, this untrained model provided comparable performance to
`cross-encoder/ms-marco-TinyBERT-L-2-v2` with finetuning. The precision was lower, but the recall higher. The other pretrained models are useful for judging

the similarity of a pair of sentences, but considering that the retrieved documents in our dataset contain more info than needed to answer the query, we expected that these models would not be useful.

3.3 Epochs

Before training the model, it is possible to enter an amount of epochs, how often the model trains on the data. We did a few tests where we trained the same model on the same data, but with a different amount of epochs. To save time during these tests, we used only the first 5000 lines of data. As expected, more epochs improved the results, but also increased the time took to train.

3.4 Input length

When choosing the model, it is possible to set the maximum input length in tokens that it will process as input. The documents are rather long: the first one in `dev\_data.csv` is over 1000 words long. One example in the SBERT documentation set the maximum amount of tokens at 256, so we initially use this. When testing an otherwise identical model with 512 tokens, the precision of the model improved, but the recall unexpectedly dropped. It also took somewhat longer to train, so we went back to maximum length 256.

3.5 Evaluators

One of the hyperparameters to use during training is the evaluation class, which SBERT uses to judge how good a model is during training. Cross-encoders have separate evaluation classes from other types of models, and the full list can be viewed in the SBERT documentation[Rei21a]. Based on this list, we have decided to test `CEBinaryAccuracyEvaluator` and `CEBinaryClassificationEvaluator`, which are meant for binary labels, and `CERerankingEvaluator`, which is more useful for the re-ranking. The other evaluators need either continuous labels or multiple outputs, which we do not use.

In our tests, the `CEBinaryAccuracyEvaluator` had a higher precision than `CEBinaryClassificationEvaluator`, but a much lower recall. `CEBinaryClassificationEvaluator` Had more balanced results for precision and recall. `CERerankingEvaluator` had very similar results to `CEBinaryAccuracyEvaluator`, with slightly lower precision and slightly higher recall.

3.6 Final model

For the final model, we decided to go with `CEBinaryClassificationEvaluator`. The full training dataset training_data.csv is quite a bit larger than the dev_data.csv, so we decided that we would only train it for one epoch, despite more giving better results in the tests. Despite this, it still took several hours to run. Testing the precision and recall of this model on dev_data.csv gave results of about 0.66 and 0.61, which is somewhat lower than the results from our previous tests, which could get results above 0.8. However, we think that the reason for the better score on smaller tests is overfitting, as we trained the model on the same data that we calculated precision and recall for.

References

[Jos21] Preetam Joshi. *RBO v/s Kendall Tau to compare ranked lists of items.* 2021. URL: https://towardsdatascience.com/rbo-v-s-kendall-tau-to-compare-ranked-lists-of-items-8776c5182899/ (visited on 12/15/2021).

[Rei21a] Nils Reimers. $Cross_Encoder--Sentence-Transformersdocumentation$. 2021. URL: https://www.sbert.net/docs/package_reference/cross_encoder.html#evaluation/ (visited on 12/15/2021).

[Rei21b] Nils Reimers. *MS Marco Cross-Encoders – Sentence-Transformers documentation.* 2021. URL: https://www.sbert.net/docs/pretrained-models/ce-msmarco.html (visited on 12/15/2021).

[Rei21c] Nils Reimers. *Pretrained Cross-Encoders – Sentence-Transformers documentation.* 2021. URL: https://www.sbert.net/docs/pretrained_cross-encoders.html/ (visited on 12/15/2021).

[Rei21d] Nils Reimers. *Retrieve and Re-Rank – Sentence-Transformers documentation.* 2021. URL: https://www.sbert.net/examples/applications/retrieve_rerank/README.html#re-ranker-cross-encoder/ (visited on 12/15/2021).

[RG19] Nils Reimers and Iryna Gurevych. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing.* Association for Computational Linguistics, Nov. 2019. URL: https://arxiv.org/abs/1908.10084.

[Tha+21] Nandan Thakur et al. “Augmented SBERT: Data Augmentation Method for Improving Bi-Encoders for Pairwise Sentence Scoring Tasks”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* Online: Association for Computational Linguistics, June 2021, pp. 296–310. URL: https://www.aclweb.org/anthology/2021.naacl-main.28.