Sentence Similarity for Internal Knowledge Bases

Matthew Cline
College of Engineering
Cornell Tech
New York, New York
Email: mc2579@cornell.edu

Jacob Everly
College of Engineering
Cornell Tech
New York, New York
Email: je354@cornell.edu

***Abstract*—We want to tackle the practical problem of outdated documentation in internal knowledge bases using sentence similarity. Specifically, we would like to find a way to match knowledge shared using messaging tools to a team's existing internal knowledge base. After running a few experiments, we learned that open-source sentence transformers can be used to successfully solve the problem. We implemented a bi-encoder sentence transformer to identify the most likely matches for a particular query. For most queries, it correctly predicted the most relevant sentence. Going one step further, we fine-tuned a cross-encoder sentence transformer with customized data for our application of interest. This extra layer ensured our model predicted the correct matching sentence from the list of matches produced from the bi-encoder sentence transformer.**

I. Introduction

In professional and educational settings, individuals share valuable information with each other via messaging tools - in many cases this information is not migrated to their team or school's internal knowledge base where it could provide a lot of value to many others. We implemented an application of machine learning to match knowledge shared using messaging tools (i.e. Slack, Teams, etc.) to a team's knowledge base (i.e. Confluence, Notion, etc.). In practice, we compare a sentence from a messaging tool to all sentences in a knowledge base and identify the most correlated sentence (and therefore document) for the input sentence.

A. Process

1) Knowledge share: Employee/student shares a message in Slack that should also be added to a team's knowledge base for future reference and to serve as contextual knowledge

2) Pass in inputs: Our sentence similarity model takes two inputs: a list of sentences corresponding to the text in the knowledge base and a single sentence that represents the original message from Slack (assuming it's just one sentence).

3) Use model to predict related sentence: The model produces a score for each sentence in the knowledge base indicating how closely related it is to the input sentence.

4) Visually pair the two sentences with a new tool: We match the input sentence with the highest scored sentence in the knowledge base returned from the model and display it as overlay text in a new tool.

This is an application that could be used by companies and universities to update their knowledge base with supplemental knowledge.

II. Background

In practice, there are two different methods that are widely used to calculate sentence similarity. One method is the bi-encoder approach in which two input sentences are independently vectorized using BERT[1] and compared by calculating cosine-similarity. This approach is best for calculating similarity in large datasets since vectorizing individual sentences and calculating cosine-similarity is very efficient.

The other approach is to use a cross-encoder. A cross-encoder passes both sentences to BERT in order to calculate a similarity score. The similarity score is more computationally intensive but generally yields more accurate results than a bi-encoder. Since the similarity score is computationally intensive, it's best to use a cross-encoder for small datasets.

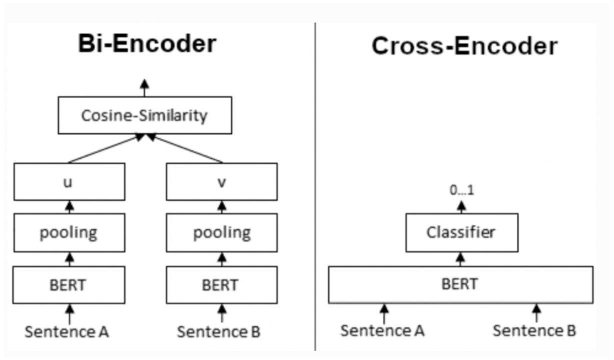

Fig. 1. Bi-Encoder vs. Cross-Encoder Diagram[8]

III. Method

We used a handful of different methods to calculate sentence similarity.

Firstly, we used the SentenceTransformers[2] library from SBERT.net to calculate sentence embeddings in a vectorized format for our baseline. To evaluate the models, we selected a few Wikipedia articles along with a few sentences in those articles to rewrite with a similar meaning. We then calculated the cosine similarity of the handful of rewritten sentences with all of the sentences in the Wikipedia articles and ensured the model selected the most relevant article and sentence.

Secondly, we used a combination of a bi-encoder and cross-encoder sentence transformer model to evaluate performance on data more closely related to the problem we wish to solve.

In order to solve for this specific application of machine learning in enterprise settings, we collected and labeled data from public community channels and corresponding documentation sources. We read through a public Slack workspace for a JavaScript library called Babel[3] consisting of a robust community of 2000 developers. We identified questions and tips that developers shared in a channel called "development" along with relevant knowledge in Babel's documentation. We then created 100 samples of these matches in the following format:

```
{
  'sentence1': "Presets run after plugins",
  'sentence2': "Presets can contain other
                presets, and plugins with
                options.",
  'score': 0.9,
  'split': 'train'
}
```

where 'sentence1' contains the message from Slack and 'sentence2' contains the relevant text in documentation. We also included a score to quantify how related each pair of sentences is and a 'split' value for separating our train and test sets. In addition to highly correlated sentences with high score values, we also included samples of unrelated sentence pairs with low score values.

In addition to Babel, we gathered data from another source called Ultimaker Forum[4], which is a community forum for 3D printers. Similar to what we did with Babel data, we matched messages from Ultimaker Forum with relevant text in product documentation. The dataset was conducted by surveying online forums, whose focus was technical troubleshooting. From the forums we could see problems and solutions relating towards physical systems. When we found a solution or problem we then looked in the documentation released by the manufacturer that could solve the problem. Through the given solution related to the forum's content, a score was then given based on how relevant the two sentences were to each other.

With this data we collected, we implemented a bi-encoder sentence transformer model layer which produces the top 32 sentence matches from the input dataset given a specific query. Our input dataset was the 'multi-qa-MiniLM-L6-cos-v1' dataset from HuggingFace, which contains 215 million question-answer pairs from diverse sources[5], along with the sentence data we collected from Babel and Ultimaker Forum.

After the bi-encoder layer selects the top 32 sentence matches for a specific query, we send those 32 sentences to the next layer, the cross-encoder sentence transformer. The cross-encoder model calculates a score between the query and each of the top 32 sentence matches from the previous layer and reorders the sentences according to its score. We created the cross-encoder from a base model called 'distilroberta-base'[6] and fine-tuned it with the data we collected and labeled.

IV. Experiments

We first ran a baseline experiment to determine the effectiveness of SentenceTransformers in predicting sentence similarity from a few different Wikipedia articles.

We loaded three different Wikipedia articles ("Cosine similarity", "Product requirements document", "Internet of Things"), split them into sentences, and selected one sentence to test the model's prediction.

```
"PRDs are most frequently written for software products, but can
be used for any type of product and also for services."
```

Fig. 2. Specific Sentence

We created a new "contextual" sentence to test if our model recognized this contextual sentence as the most similar sentence to the original sentence. We wrote this sentence in a way that provides contextual information about product requirements documents (PRDs) and resembles text one might send in a conversational tool like Slack.

```
"Most software products require PRDs because they provide a
standardized way to document the features and constraints of a
product."
```

Fig. 3. Contextual Sentence

We added this new sentence to the list of sentences from all of our Wikipedia articles. Then, we used the SentenceTransformers library to encode the sentences into text embeddings, which are represented as n-dimensional vectors. We then used those embeddings to calculate cosine similarity between our original sentence and all other sentences across the Wikipedia articles along with our contextual sentence we added. When we predicted the most similar sentences to the original sentence, it correctly predicted the contextual sentence.

After the baseline experiment, we ran a more robust experiment with our collected data using the bi-encoder and cross-encoder sentence transformer models. We trained our bi-encoder transformer with all of the sentences we collected and fine-tuned our cross-encoder with 82 of the labeled pairs of sentences, holding out 18 for testing.

Initially, we fine-tuned the cross-encoder with only the labeled data points we created. After seeing low accuracy on our test dataset for the cross-encoder (Pearson correlation score of .4), we decided to supplement our fine-tuned dataset with the STS Benchmark dataset, which contains semantic semantic textual similarity data from ”image captions, news headlines, and user forums”[7], we noticed a much higher Pearson correlation score of .83 for a test set containing 18 of our own test samples and 1379 samples from the STS Benchmark dataset. In addition to the improved test accuracy score, we also noticed better, more intuitive sentence matches from the cross-encoder. The cross-encoder model fine-tuned with only our labeled data had too small of a sample size which produced less accurate results. The solution was to leverage existing labeled data in addition to the data we collected to improve accuracy.

We noticed that for most queries, the bi-encoder was sufficient for predicting the most relevant sentence match, i.e. it produced the same results as the cross-encoder. However, we did notice examples where the cross-encoder gave a higher score to a more relevant match than the bi-encoder. Here’s an example of this case:

```
>>> search(query="What are Babel presets?")

Top-3 Bi-Encoder Retrieval hits:

0.663    is there a way to configure
         plugins used by babel-preset-env
0.649    @babel/preset-env takes any target
         environments you've specified and
         checks them against its mappings
         to compile a list of plugins and
         passes it to Babel.
0.547    @babel/types

Top-3 Cross-Encoder Re-ranker hits

0.917    @babel/preset-env takes any target
         environments you've specified and
         checks them against its mappings
         to compile a list of plugins and
         passes it to Babel.
0.894    is there a way to configure
         plugins used by babel-preset-env
0.659    Presets can contain other presets,
         and plugins with options.
```

In this example, there is one sentence that provides the most clear answer to our input query, and that’s the sentence the cross-encoder ranks as highest and the bi-encoder ranks as second. This provides evidence that using a cross-encoder layer after the bi-encoder layer as an extra filter can yield better results. Since we’re only calculating a finite number of sentence similarities in the cross-encoder layer, as opposed to the bi-encoder layer which calculates sentence similarities for all sentences in the large dataset, this extra layer of computation is not time-intensive and therefore worth the improvement in results in practice.

V. Discussion and Prior Work

Our baseline experiment proved that sentence transformers successfully match sentences from Wikipedia data. After collecting and experimenting with public data that is closer in nature to the problem we are trying to solve, we learned that we can just as effectively match sentences from enterprise and educational settings using bi-encoder and cross-encoder sentence transformers.

VI. Conclusion

In summary, we believe that open source sentence transformers are powerful enough to help us solve the problem of outdated documentation for enterprise and educational settings. The biggest challenge is to gather enough high-quality data from public sources. In addition to Slack and online community forums, we plan to gather data from Discord communities since Discord is a very popular tool for developers.

In addition to documentation, we also want to experiment with mapping messages from messaging tools with code. For example, we believe there’s value in mapping questions and answers from Slack to code in Github pull requests to provide additional context.

We believe this application of machine learning will improve productivity for employees and students and lead to more efficient and cohesive teams.

Acknowledgment

The authors would like to thank Professor Volodymyr Kuleshov for his guidance regarding experimentation and data collection.

References

[1] *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, https://arxiv.org/abs/1810.04805
[2] *SentenceTransformers*, https://www.sbert.net/
[3] *Babel*, https://babeljs.io/
[4] *Ultimaker Forum*, https://community.ultimaker.com/
[5] *multi-qa-MiniLM-L6-cos-v1*, https://huggingface.co/sentence-transformers/multi-qa-MiniLM-L6-cos-v1
[6] *distilroberta-base*, https://huggingface.co/distilroberta-base
[7] *STS Benchmark*, https://paperswithcode.com/dataset/sts-benchmark
[8] *Bi-Encoder vs. Cross-Encoder*, https://www.sbert.net/examples/applications/cross-encoder/README.html