Identifying Hard Negative Topics via Bad Annotation Detection

Antonín Jarolím xjarol06
Vojtěch Eichler xeichl01
Ondřej Vlček xvlcek27

Abstract

In the realm of Czech language understanding by Large Language Models (LLMs), a notable gap exists in the absence of a robust benchmark for evaluating comprehension. This project aims to address this void by proposing one benchmark task to select a topic which occurs in the text from a predetermined set of topics. Leveraging a dataset annotated with positive topics, the objective is to add hard negative examples to ensure task complexity. We use two methods to generate hard negatives A) Using topics from most similar texts in the dataset. B) Asking LLMs directly for hard negatives. Then, in both of these methods, we use a bad annotation detector to remove completely unrelated topics. This approach results in two lists for each text. First has the benefit of preserving a clear dataset (as topics are from annotators) which would not bias benchmark tasks towards models which generated topics, and second generates harder negatives. The entire codebase for this project is available on Github[1].

1 Detecting bad annotations

Datasets are the core of machine learning and it's essential for them to be correct and representative of the task. The texts in SemAnt dataset are being annotated by humans and as we know humans make errors. In this part of the project, we focused on annotations of topics for texts. We attempted to find a good approach, to be able to detect these errors. Such detections could be used to flag the annotation so that the control of annotations is easier and the attention of the controller is aimed at potential flaws as well as to discard obvious errors in annotations. As of May 2024, we weren't able to find solutions for this problem, which was surprising as we believe there's real value in such a tool.

1.1 Cosine similarity scores

The obvious approach is to create embeddings of the text and all of the topics as well. These embeddings should capture the semantic meaning of text inputs. This allows us to compare each topic embedding with the embedding of the text using cosine similarity. We used these similarities to predict, whether a topic is relevant for the text or not and how confident are we about it. As it is the case with all approaches we experimented with, it's possible to fine-tune the threshold which would suggest an error in annotation. This would allow us to either filter out really bad annotations and miss annotations that might be on the edge of being an error, or in the other case filter/flag many annotations where there's at least a bit of suspicion that the annotation is an error. As we're dealing with texts in Czech, it's important that the embedding model was trained on Czech as well. The advantage of this approach is its simplicity and the fact that as embedding models get better, the results of this approach should also get better.

1.2 Asking LLM directly

The next approach we tried is again quite obvious and was suggested to us by Ing. Beneš and dr. Hradiš independently on two separate occasions. This approach aims to use the power of LLM such as chatGPT to score the annotated topics directly, by providing both the text and topics to chatGPT and asking it to score the relevance of each topic to the text. The advantage of this approach is definitely it's simplicity,

[1] https://github.com/Ajchler/knn-fit

but this comes with few cons as well. First of all, this approach comes with the additional cost of OpenAI API calls. The other concern is, that if the dataset was supposed to be used to evaluate LLMs, using one of them to clean the data would inherently result in a bias. The impact and strength of the bias is a question that is not explored in this project but should be kept in mind.

1.3 Using LLM generated topics

The last, most complicated approach to detect bad annotations is to use LLM to generate relevant topics to the text and use them for comparison with annotator topics. The idea is, that whenever none of the LLM-generated topics is similar to the annotator topics, then the annotation is incorrect. We used two methods – the first is to compare each generated topic with each annotator topic and select maximum of these values. We refer to this approach as one-to-one. Second, we concatenate all generated topics into one string and then compare it to each annotator topic. To compare them, we used cosine similarities and a pre-trained Cross-Encoder. The advantage of using cosine similarities is, that each text is passed to the model only once. An alternative way to obtain similarity scores is to use a Cross-Encoder, which requires passing the pairs of text and topic, changing the complexity to n^2. However, it can achieve higher performance. In our task, there are not many combinations of texts and topics, therefore the higher complexity is not an issue [2]. We used Cross-Encoder [3] pre-trained on the Quorra Duplicate Questions dataset as a proof of concept.

2 Generating Hard Negatives

To generate hard negatives, we tried two approaches. The first one is straightforward – we ask LLM to generate hard-negatives directly. However, it turned out, that this approach has its challenges. The other one is finding hard negatives by finding the most similar texts in the corpus and taking their topics as hard negatives for this text. This approach is motivated by an effort to obtain more ”clear” non-LLM generated topics, which would not bias the dataset towards any LLM.

2.1 Asking LLM to Create Hard Negatives

In this section, we delve into the concept of utilizing LLMs for generating hard negatives. All proposed methods were used with GPT-4-Turbo. GPT-3.5 was also tried, but proved insufficient for the task, as it failed to provide relevant data or accurately formatted output. Our approaches can be split into the three following groups.

2.1.1 Direct generation

First, we have tried the simplest approach – direct generation of hard negatives from an input text. GPT was provided with a description of hard negatives and was tasked with creating a fixed number of them for a given text. Outputs for this method were consistent negatives in respect to the input text, but were also broadly inaccurate or irrelevant, making them poor candidates for hard negatives.

It was more reliable to tell GPT, that it was trying to fool an opponent, instead of defining what a hard negative was. This opponent was supposed to be either a human or a smart language model. It was then instructed to generate topics which seem to describe the text but don't. To further anchor the generated negatives closer to the input text, GPT was instructed to follow the annotated topics. Telling the model to provide a list of explanations as to why its topics met the specified criteria also seems to reduce the number of completely irrelevant outputs.

Topics generated by this approach were far more related to the input text, making them better candidates for hard negatives. This also led to a potential drawback – many negatives were too pedantic and relied on expertise in the subject area of the text. This can however be a desirable trait depending on the

[2] The advantages of Cross-Encoder are described here.

[3] Cross-encoder model on huggingface.

purpose of the hard negatives. A related issue with this method is that some false negatives tend to be included only because they are considered too specific to one section (and therefore don't describe the text as a whole) or too broad. Another major drawback is that using annotated topics makes this method reliant on their quality. Inaccurate annotations would almost always lead to unusable negatives.

Alternatives aimed at creating slightly misleading descriptions or correctly describing one section of the text but not another performed similarly with the same downsides, except for being independent of annotated topics.

The best results were, in our opinion, from a more relaxed approach. It was inspired by the idea, that hard negatives should come from the same context as the text. To explore this context, GPT was prompted to simultaneously generate topics describing the text and to provide alternatives to these topics. These alternatives should represent a similar concept and should be from the same context as the original topic. They also strictly shouldn't describe any part of the text. The resulting negatives are far more diverse than with the previous approach while keeping a connection to the original text. It still suffers from false negatives, however.

2.1.2 Word focus

A far less successful approach was to focus on individual words. The idea behind this approach was that an AI might fixate on important words within the text, incorrect topics based on those words could therefore make for good hard negatives. The major flaw with this method is that GPT-4 will tend to make up brand-new word combinations, even when explicitly told not to. Our favorite examples of this include "morální dřevo" or "lidská flóra".

To combat these novel combinations, GPT was instructed to use word pairs from the text instead. This improved the topic relevance but had a new flaw – most neighbouring words weren't reasonable topics, e.g. a verb and a noun. Allowing GPT to choose a pair of arbitrary words merely combined the flaws of this method and the one in the previous section.

2.1.3 Two-phase approach

Two-phase approaches were also tried as a form of task decomposition. One was to generate hard negatives in a brute-force manner – first, GPT generated a list of topics loosely describing the input text. Then, in a separate conversation, it was instructed to keep those, which did not describe the input text (keeping both in the same conversation naturally led to GPT claiming, that all topics were accurate). This produced some hard negatives but was generally inefficient, averaging roughly one hard negative out of twenty generated topics. This was caused by the fact that the second phase chose either clear negatives (mistakes from the first phase) or incorrectly chose positives. A second major flaw was that the model often outputs synonyms due to the large number of topics generated.

A second approach was to first generate pairs of similar descriptions – one for a section of the text and one for the text as a whole, the idea being similar to that of the *word focus* section. The second step was to find a topic representing the segment description, but contradicting the broader one. GPT, however, could make this very difficult by choosing vague descriptions or topics.

Self-reflection was also attempted to improve the quality of generated topics. GPT was prompted to reconsider its previous output and the initial criteria as a second part of the same conversation. This resulted in fewer irrelevant topics but also often changed negatives into positives.

2.2 Finding Hard Negatives in Dataset

We first create embedding for each text in the dataset. Then, we find the most similar texts in the dataset by computing cosine-similarity scores. For each text, we take top-20 similar texts by this metric and union all topic sets for this text. Then, we remove all annotator topics from this set. We used sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2 model to obtain embeddings for this task.

3 Using Bad Annotation Detection on Generated Hard Negatives

Both approaches to generate hard negatives have some examples that are way too easy, meaning they are not really hard negatives. Especially the approach which is finding the most similar texts produces a lot of easy examples. The goal of this task is to remove them to create smaller more precise sets. In our experiments, we found out, that one of the best approaches to detect bad annotations is cosine similarity. As it does not cost any money, we decided to use this approach for this task.

We tried two methods. First is, to sort a set of potential negatives by **highest cosine similarity** and select only the top 5 of them. This leads to a set, which has a few positive topics, however it also contains all of the possibly hardest negatives. The second one is based on evaluation from experiments. We found out, that threshold 0.4 leads to the best cDet score, therefore, the idea is to preserve only topics, whose **cosine similarity is closest to this threshold**. We once again selected the top 5 topics based on their proximity to the established threshold.

4 Evaluation of Experiments

In order to evaluate all of our approaches we needed a dataset, which would have labels for annotations which would label the topics as either correct or incorrect. Since we didn't have such a dataset, we had to create our own. We created a simple CLI tool for annotating topics, which can be found in `utils.py` as a `GoldDatasetCreator` class[4]. We will refer to this dataset as golden dataset[5] in the rest of this report. The golden dataset contains 35 texts and more importantly 38 correct topics and 21 incorrect topics spread amongst the texts. It's important to keep in mind that this dataset was created and checked by the three of us and since we all discussed the problem together on regular basis, it might contain errors, even though we did our best to make sure the labels are correct.

4.1 Evaluation of Bad Topics Generation

In Figure 1, there is an evaluation of the following models/approaches to detect bad topic annotations:

- Comparison of text and topics by cosine similarity of embeddings. We tried multiple models which were trained on Czech texts: **cos-sim-score-LaBSE**: Google pretrained BERT multilingual model for sentence embeddings. **cos-sim-score**: Paraphrase Multilingual Sentence trasformer model sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2 and **cos-sim-score-base-v2**: Larger version of this model sentence-transformers/paraphrase-multilingual-mpnet-base-v2. And **cos-sim-score-googlebert-cased**: another BERT-based google model google-bert/bert-base-cased.
- **direct-score**: Prompting LLM (gpt-4-turbo) for direct score [6].
- Comparison of generated topics with annotator topics. **gen-CE**: Concatenated list of generated topics compared with annotator topics using Cross-Encoder. Comparing generated and annotator topics one by one using Cross-Encoder (**gen-CE-1to1-max**) and cosine similarity (**gen-cos-sim-max**).

We also found mincDet score ($(fnr + fpr)/2 * 100$) for all approaches. To our surprise, the smallest value we found is $minCdet = 17.669$ for **gen-ce**. Right after it, there is **cos-sim-score** with $minCdet = 18.734$. The mentioned approaches also have a nice smooth curve in the DET curve in

[4] https://github.com/Ajchler/knn-fit/blob/98e262084b61316e4f1e8cfa36973977ab2997f6/utils.py#L109

[5] Gold annotated dataset on GitHub.

[6] Prompting LLM for direct score.

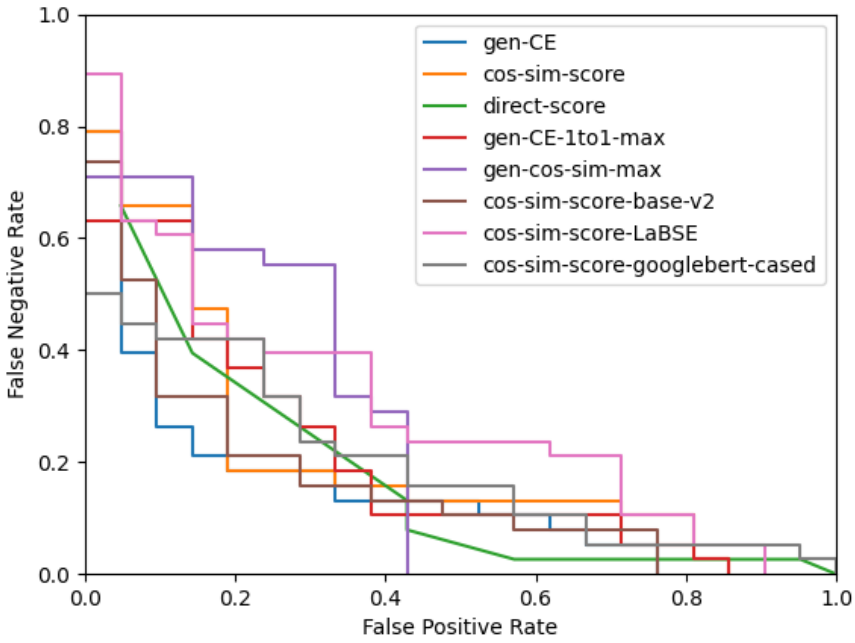

Figure 1: Evaluation of bad topics generation on DET curve. Note that axes are not scaled non-linearly by standard normal deviates.

Figure 1. Determing the smallest cDet score also effectively leads to the finding of the best threshold for each approach. We used threshold for **cos-sim-score** of 0.4 to find the best hard-negatives, as described in the previous section. In the DET curve, one can observe the difference between **cos-sim-score** and **cos-sim-score-base-v2**. For the second one, the curve is more smooth, which it's beneficial facilitate operational adjustments of the threshold.

4.2 Evaluation of Hard Negatives

The evaluation of hard negatives is even more challenging than evaluating bad annotation detection. We thought of an approach where these hard negatives would be part of the classification task and we would somehow measure, how many models would get it correct as negative, but this would require a lot of effort and probably many carefully chosen models, we believe that defining such evaluation method would be very complicated and is out of the scope for this project, as it might be complete nonsense as well. Therefore we decided on qualitative evaluation in which we merged the potential hard negatives from all approaches for the golden dataset texts in one file: `evaluation-data/merged_hard_negatives.json`.

For the two-phase approach we believe we would require much bigger text corpora as most of the topics seem very random. It's important to believe that we are probably working on a small subset of the actual dataset (we only managed to scrape around 6k texts) and the texts are diverse. On the other hand, even this approach managed to find some potential hard negatives and it would be interesting to see how this approach does when more data is available.

The generated potential hard negatives seemed more promising, not only did we feel like they stayed closer to the actual topics, but we'd have to take a moment and really think about whether it is negative or positive. That being said, this approach also produced many topics which we thought were either obvious negative or obvious positive. We believe that bad annotation detection and its counterpart good annotation detection, which would be easy to implement, could filter some of these obvious non-hard topics out and therefore make it potentially better at providing potential hard negatives. We did not manage to experiment with this as we didn't have enough time since we were playing with the generation

basically until the deadline and only came up with the filtering idea near the end of the project.

It's hard to say, whether this approach could be used fully automatically, because of the obvious negatives/positives as mentioned. We suggest this approach could at least be used as an inspiration for the annotator trying to come up with hard negatives, or even could be used as potential hard negatives, where the annotator only has to remove the few remaining obvious ones.