Chunking Evaluation Report

Artem Abaturov

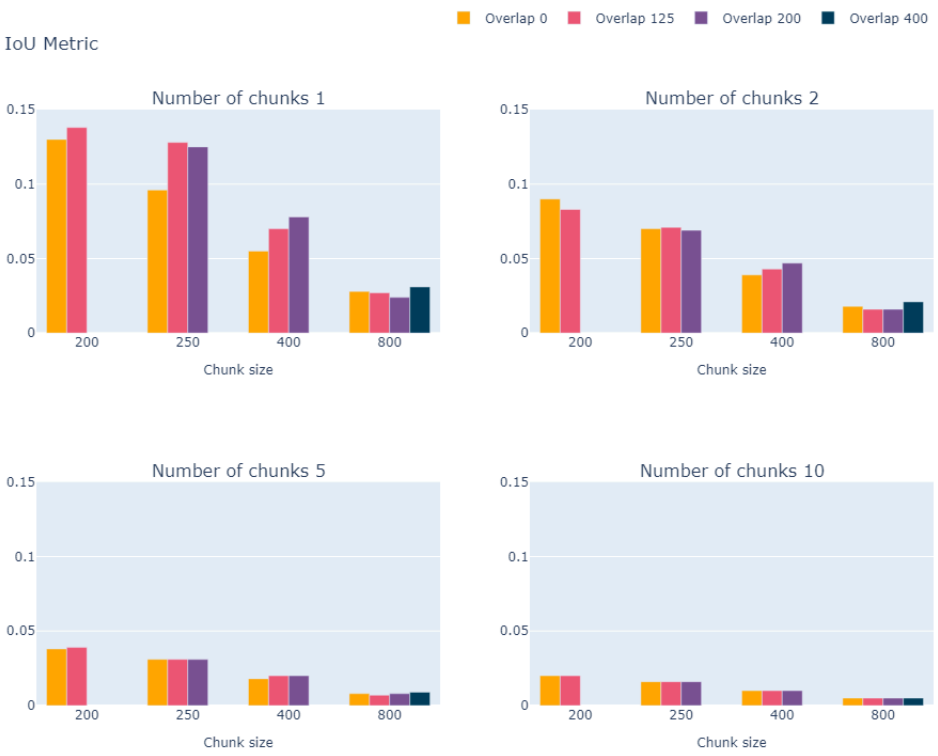
1. Implementation Details

Firstly, I would like to try to explain my implementation. All code files are in *src* directory, and all data is contained in *data* folder.

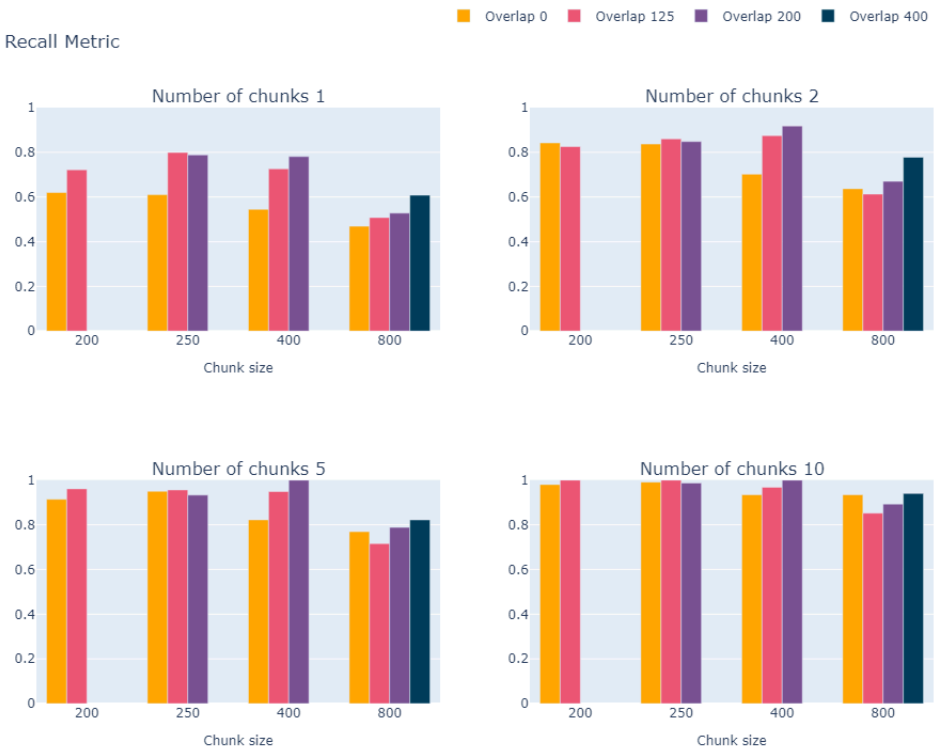
- *data_loader.py* contains two functions: the first returns corpus by its name, and the second returns dataframe with all questions related to the corpus with the specified name.
- *fixed_token_chunker.py* is the implementation of TokenText chunker from the paper.
- *embedding.py* contains the function *generate_embeddings* which generates embeddings for all given texts.
- *retriever.py* contains the class *Retriever*. It has *vector_store* field and the function *retrieve*, which takes *query* and *top_k* and returns *top_k* most similar chunks on *query*. As a distance, I use cosine similarity.
- *evaluation.py* contains the class *Evaluation*. It has the fields *corpus*, *questions*, and *chunker*. One interesting thing is that in the paper, precision and recall are defined using tokens, but in the implementation, they are defined using symbols. I decided to follow the same approach. The class has two functions: *calc_query_metrics* and *calc_metrics*. The first one calculates IoU, precision, and recall for a given question_id and the retrieved chunks. The second one calculates the mean IoU, precision, and recall for all questions; it uses the first function. It also calculates the standard deviation for all metrics.
- *pipeline.py* contains only one function *run_pipeline* which takes a configuration and returns metrics for our chunker. You can check the format of the configuration in this file.

2. Charts

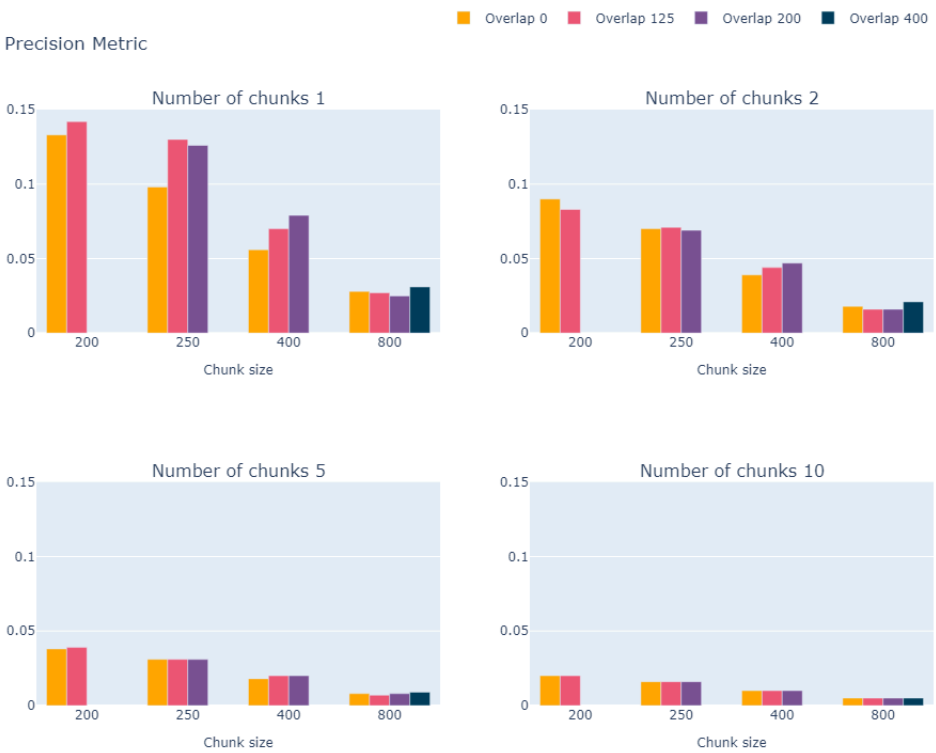
In the first graph, we see the dependency between IoU score and chunk size with overlap. With a chunk size of 200, we achieve the maximum IoU score, after which it decreases. Also, we can see that the IoU score does not always depend linearly on the overlap size.



In the second graph, we naturally see that the number of chunks and overlap increase recall.



Based on the third graph, we can conclude that the size of chunks decreases precision. It's quite clear because we just retrieve more useless tokens.



3. Key findings

- Recall for chunk size 800 is less than for 400. It's interesting because with chunk size 800, we retrieve more tokens and, therefore, we expect a larger recall. One possible explanation of this effect is that our embeddings work worse with big chunks, and our retriever returns a fewer number of useful chunks.
- In the configuration *chunk_size*=250, *overlap*=125, *number_of_chunks*=1 we have recall equal to 0.8 which means that on average only one chunk covers 80% of useful tokens which means that the correct information for our queries lie closely.
- The optimal configuration from my side is the following: chunk size is 400, overlap is 200, and number of chunks is 5. This configuration achieves recall 1 and has a precision bigger than configurations with bigger chunk sizes. From my experience, I consider recall more important than precision in this problem.