8월 30일

SentenceBERT를 통해 Reranking하는 클래스

```
class QueryDocumentRanker:
    def __init__(self, qa_context, DPR, BM25,top_n):
        self.qa_context = qa_context
        self.DPR = DPR
        self.BM25 = BM25
        self.model = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-12-v2',max_length=512)
        self.top_n = top_n
        for i in range(len(self.DPR)):
            random.shuffle(self.DPR[i])
            random.shuffle(self.BM25[i])

    def rank_documents(self):
        total_scores = []
        for i, query in enumerate(self.qa_context):
            combined_result = list(set([(idx, paragraph) for idx, paragraph in self.DPR[i] + self.BM25[i]]))
            random.shuffle(combined_result)

            # Calculate similarity scores using the model
            scores = self.model.predict([(query, doc_text[1]) for doc_text in combined_result])

            # Combine the scores with the corresponding paragraphs
            # combined_scores = list(zip(combined_result, scores)) # score까지 챙기는 경우
            combined_scores = list(combined_result)

            # Sort the combined_scores by score in descending order
            combined_scores.sort(key=lambda x: x[1], reverse=True)

            # Print the top 3 documents for the current query
            # print(f"Query {i + 1}: {query}")
            # for j, ((idx, doc_text), score) in enumerate(combined_scores[:3]):
            #     print(f"Top Document {j + 1} (Index {idx}):\n{doc_text}\nScore: {score}")
            #     print("=" * 30)
            total_scores.append(combined_scores[:self.top_n])
        return total_scores
```

Reranking하면 (index, sentence)가 있는데 여기서 index는 Cycle_passage의 index를 의미한다.

그렇다면 이 클래스를 Reranking for looking sentence로 하고 Reranking 결과를 통해 바로 cycle passage를 선별하는 과정을 거치는 클래스를 만들자.

```
class QueryDocumentRankerForIndex:
    def __init__(self, qa_context, DPR, BM25, top_n):
        self.qa_context = qa_context
        self.DPR = DPR
        self.BM25 = BM25
        self.model = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-12-v2', max_length=512)
        self.top_n = top_n
        for i in range(len(self.DPR)):
            random.shuffle(self.DPR[i])
            random.shuffle(self.BM25[i])

    def rank_documents(self):
```

```
        top_results = []  # List to store the top results
        for i, query in enumerate(self.qa_context):
            combined_result = list(set([(idx, paragraph) for idx, paragraph in self.DPR[i] + self.BM25[i]]))
            random.shuffle(combined_result)

            # Calculate similarity scores using the model
            scores = self.model.predict([(query, doc_text[1]) for doc_text in combined_result])

            # Combine the scores with the corresponding paragraphs
            combined_scores = list(zip(combined_result, scores))

            # Sort the combined_scores by score in descending order
            combined_scores.sort(key=lambda x: x[1], reverse=True)

            # Append the indexes from combined_result to top_results
            top_indexes = [item[0][0] for item in combined_scores[:self.top_n]]
            top_results.append(top_indexes)

        return top_results
```

sentenceBERT를 통해 나오는 결과를 cycle_Triple의 index로 나오도록했다.

```
def selected_cycle_triple(cycle_triple, top_N_ReRankindex):
    result_cycle_triple = []

    for i, indices in enumerate(top_N_ReRankindex):
        retrieved_sublists = [cycle_triple[i][idx] for idx in indices]
        result_cycle_triple.append(retrieved_sublists)

    return result_cycle_triple
```

나온것으로 cycle_triple을 선별한다.

cycle_triple로부터 edge_index pruning(graph pruning)

내일 할 일

edge_index, edge_type, cycle_list 예시를 만들어서 graph pruning 진행하기