Task 2 *(optional)*: Search Improvement

Codebase link: https://github.com/iamcalledayush/task-2

How to run: Install the following:

pip install open-ai

Run the corresponding files for the given desired techniques. For each technique I have listed the files to run.

I have divided this report into 3 sections. First, what technique worked. Second, what all techniques I tried. Third, what all techniques can be tried (proposed techniques) but will take more time.

For simply running the technique that worked, you can just run the evaluate_models.py file and it will give you the evaluation results for all 9 models (8 previous models and 1 new model that I used). I have modified the model_registry.py file to include the new model I used.

Technique that worked:

I used a different local embedding model: "**intfloat/e5-large-v2**". It is Instruction-tuned, and resulted as top-tier for semantic search as compared to default 8 available models.

The previous best model was: ***stella-base-en-v2.*** ***The metrics for this model were as follows:***

- Top-1 Similarity: 0.8100
- Weighted Top-5 Similarity: 0.7873
- Coverage: 100%
- Avg. Results: 5.00
- Response Time: 0.0213 sec

The metrics for this new model **intfloat/e5-large-v2** **are as follows:**

- Top-1 Similarity: 0.8588
- Weighted Top-5 Similarity: 0.8457
- Query Coverage (%): 100.00

- Average Number of Results: 5.00
- Average Response Time (s): 0.0173

This new model beats the previous best model in every aspect. The embeddings quality are clearly better here for this model.

Results are present in the ***evaluation_metrics_e5.csv*** file. For code, we can just run for this single model using the ***task2_e5.py*** file. Also, *I have modified the* ***model_registry.py*** *file to use this model by default along with other 8 models as well.*

Techniques I tried:

A summary of various techniques that I tried:

- *Query Paraphrasing*: I tried various paraphrasers like, T5-based model, Gemini 2.0 Flash, and state-of-the-art GPT-4o through its API. The results after paraphrasing were not great. Due to limited time, there are various other things like advanced prompt engineering techniques (eg: step-back prompting, self-refine prompting, etc.) that could have been tried.
  Results are present in the evaluation_metrics_transformed.csv file and code is present in task2_gpt40.py file.
- *Combining GPT-4o API with OpenAI's embedding model "text-embedding-ada-002":* The results were the best with this approach, but it lacked behind in the response time metric:
  - top-1similarity: 0.866
  - weighted top-5 similarity: 0.854
  - coverage percent: 100
  - avg number of results: 5
  - avg response time: 0.51549

  The improvements in first two metrics are not significant enough as compared to the **intfloat/e5-large-v2** model, and the response time is significantly large. Also, there is an addition OpenAI API cost too. Hence, this approach was discarded.

  Results are present in openai_eval_summary.csv file and code is present in task2_openai.py file. I have publicly used my OpenAI API in code for now to allow anyone who wants to run my current code, but it can be used as an env variable for privacy.

- *Creating an enriched database using GPT-4o API:* I created a new database of items with additional columns like, category (category of the item), intended_use (a brief description of intended use of the item), indoor_or_outdoor (where is the item used in general; indoor or outdoor), portable (is the item portable), size (either small or medium or large). This approach was promising and could work too if we try advanced prompting techniques with GPT-4o and use various permutations and combinations of the proposed new columns and various models. This is another technique that can yield promising results. But for now with limited options, I could not experiment with a lot of combinations.

  Results are present in the evaluation_metrics_enriched.csv file and code is present in the create_new_db.py file (to create new DB) and evaluate_models_extra_features.py file (to evaluate models). I created an updated item_db_search_engine_updated.py file.

Proposed Techniques:

1. **Prompt-based Query Expansion:** We can use an LLM like Gemini or GPT-4o to rewrite queries with added synonyms and related terms to improve recall without changing intent.
2. **Hybrid Query:** We can generate multiple versions (say, original, paraphrased, expanded), compute embeddings, and average them for a richer query vector. As a rich query vector is really important for attention-based models.
3. **Query Classification:** We can use classification algorithms to classify each query into a predicted item category (e.g., furniture, electronics) and only search within that category to reduce noise and improve performance/response-time.
4. **Fine Tune the current best (E5) model:** This approach is taking long as E5 is a large model and takes a long time to get fine-tuned. But this method is promising with respect to improving the performance and accuracy of search.
5. **Multi-**Agentic RAG system: We can use a couple of LLMs to create a RAG pipeline to perform the search using advanced prompting techniques like: Self-Refine prompting, Self-Consistency prompting, AlphaCode 2's re-ranking method based prompting, and Step-Back prompting.