SI 650 Final Project Report

Wenfei Tang

Dec 12, 2022

Introduction

In this project, I would like to explore how we can build an information retrieval system which can accomplish a retrieval task where real-world commonsense reasoning ability is needed in this task. Specifically, given an action language description as a query, our goal is to build a retrieval system which retrieves the relevant effect texts describing the effect states of the query action. For example, given the action quey “someone slices the potato”, “the potato becomes into pieces” is the textual information we are interested in retrieving.

In fact, this is a objectively hard task for existing retrieval system, as the query may not seem directly related to the ideal documents to retrieve. For example, “someone slices the potato” and “someone bakes the potato” are two very similar queries, but the desired retrieved effect texts can be completely different. In this case, “someone slices the potato” emphasizes on the “in pieces, not as a whole” state, while “someone baked the potato” emphasizes on the “edible” or “is crispy” states.

Recent advancement in large pre-trained language models has made such a retrieval system with reasoning ability possible. These large language models usually have some understanding of the physical world given its enormous model size and amount of pretraining data. Zero-shot prompting from these pre-trained LMs is proven to be one useful technique to generate new knowledge statements.

The main contribution of this project are the two proposed techniques to improve the baseline model performance of the retrieval task requiring commonsense reasoning. A query expansion technique using zero-shot prompting with ChatGPT is introduced and the pre-trained large language model extracted from the CLIP bi-encoder model is utilized to compute the similarity score as well.

Provided that the baseline bm25 and tf-idf models barely outperforms the random performance model, the final model outperforms the random performance baseline model greatly in nDCG@5 and nDCG@10, and achieves an nDCG@5 score as high as 0.97.

Data

** Note: this part is different from the submitted project update report, please read it **

To solve this problem, we first need to identify a physical commonsese reasoning dataset which contains information about both the action language and the effect states description. After obtaining the data and manually annotating the queries, we would like to build a text retrieval system which can retrieve the effect text descriptions given the action text query.

The paper *What Action Causes This? Towards Naive Physical Action-Effect Prediction* (Gao et al., ACL 2018) proposes a vision-language dataset on naïve physical action-effect prediction. In this dataset, each action language is templated as a verb-noun pair. They collected 4163 images (both positive and negative) and effect language descriptions for 140 verb-noun pairs. For each verb-noun pair, we have annotated data for both effect images and effect texts. *Figure 2* and *Figure 3* show the sample images and texts from this dataset. This dataset will be used as the source data for my project.

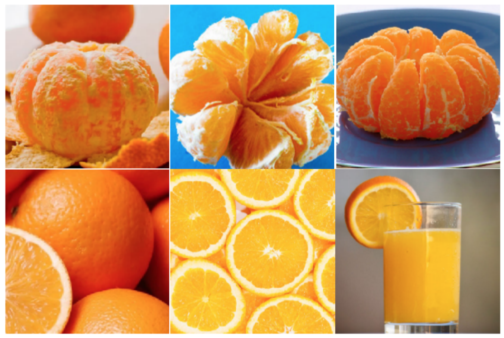

Figure 2: Sample images in the dataset of the action "peel-orange". The first row shows positive images, and the second row shows negative images of this action language.

Action	Effect Text
ignite paper	The paper is on fire.
soak shirt	The shirt is thoroughly wet.
fry potato	The potatoes become crisp and golden.
stain shirt	There is a visible mark on the shirt.

Figure 3: Sample effect texts for different actions.

To obtain a collection of documents for building the retrieval system, I preprocess the effect text data from text data in this action-effect dataset. First, I use the effect text descriptions as the document text field. Each document is given a unique string as its docno, and the corresponding action language (verb+noun pair) is stored as the metadata of this document. There are 140 documents in total**.** **Different from the project update report,** I did not use the metadata here to automatic annotate the data, and it is instead used to compute the CLIP similarity feature score used later. Besides, I also concatenated the effect text for the same action language. Now each document is much longer, consisting of 10-20 sentences. This helps reduce the annotation size from 196,000 to 19,600 and makes human annotations possible. See *Figure 4* for the document dataframe.

I generated the queries (i.e. topics) from all the action language (i.e., verb + noun pairs) by formatting them using a hand-engineered template. There are 140 queries in total. See *Figure 5* for the queries dataframe.

	docno	text	effect_vn
0	0	chairs are moved around in order. the chairs a...	arrange chairs
1	1	the flowers are in a pretty design. flowers ar...	arrange flowers
2	2	i put a potato in the oven to bake it. the pot...	bake potato
3	3	the eggs are stirred. the eggs are scrambled. ...	beat eggs
4	4	my knee needs surgery. a knee is folding in tw...	bend knee
...	...	...	...
135	135	the tree looks festive for the holiday. the tr...	trim tree
136	136	the hair is turned around on itself. the hair ...	twist hair
137	137	the rope is being contorted. the rope is being...	twist rope
138	138	the book is concealed behind wrapping paper to...	wrap book
139	139	the box is covered with something. the box is ...	wrap box

140 rows × 3 columns

Figure 4: Preprocessed collection of documents dataframe.

Given 140 queries and 140 documents, this leads to a total of 19,600 annotations, see *Table 1* for the annotation dataframe (i.e. qrel). **Different from the project update report,** I change the relevance score scale from 3 to 5 and did human annotations. Score 5 is used for perfect match. Only a perfect match in the action language is a score 5. Score 4 is used for very relevant, there are many possibilities, say if the verbs used in the query are similar (e.g. bake, boil, cook) and the nouns appeared in each query are an exact match, it is a score 4; or if the verbs are an exact match, and the nouns are of the same category (e.g. bowl, bottle, cup), it is also a score 4. Score 3 is used for somewhat relevant, if neither the verbs nor the nouns are a match, but they are somewhat related to each other, this is a score 3. Score 2 is used for a bit relevant, say if the nouns are somewhat similar, but different actions are performed, that is a score 2. Score 1 is used for not relevant at all.

I have also **re-plotted** the relevance score distribution and counted the number of appearances for each label. See *Table 1* for instances statistics and *Figure 6* for relevance score distribution. Even though the number of appearances for relevance score 1 is quite imbalance, unfortunately those pairs have no connection or relevance at all.

	qid	query	query_vn
0	0	Someone has arrange chairs.	arrange chairs
1	1	Someone has arrange flowers.	arrange flowers
2	2	Someone has bake potato.	bake potato
3	3	Someone has beat eggs.	beat eggs
4	4	Someone has bend knee.	bend knee
...	...	...	...
135	135	Someone has trim tree.	trim tree
136	136	Someone has twist hair.	twist hair
137	137	Someone has twist rope.	twist rope
138	138	Someone has wrap book.	wrap book
139	139	Someone has wrap box.	wrap box

140 rows × 3 columns

Figure 5: Preprocessed query dataframe.

Relevance score	1	2	3	4	5	Total
# Instances	15886	1746	718	1110	140	19600

Table 1: Relevance score instance count.

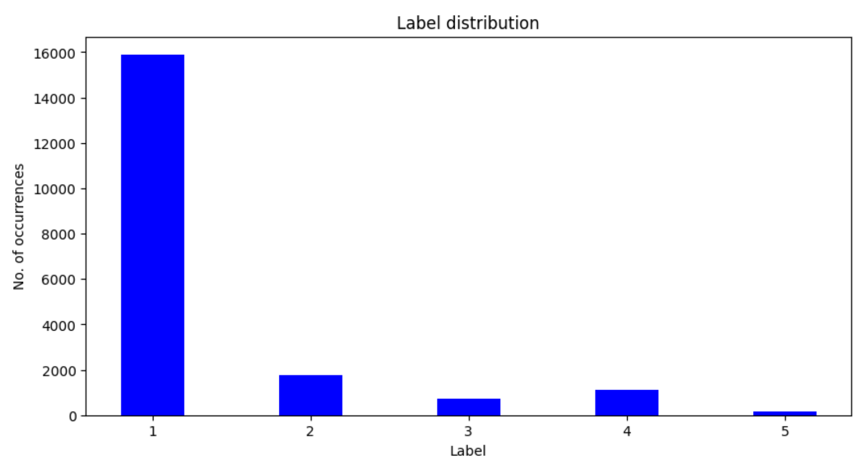

Figure 6: Relevance score distribution.

Related Work

With the development of transformer-based large language models, many pre-trained language models have come out in the recent years. These multimodal models have learned both language and vision representations through scaled pretraining steps, and already have a good understanding of both language and vision. These models can be potentially used for the cross-modal retrieval (image-text retrieval or text-image retrieval), which is related to the second part of work in this project, where we would like to retrieve a related effect state image given a action language text query. These multimodal models usually consist of two encoders, the image encoder and the text encoder, and we can make use of these pretrained encoders and their alignment for our retrieval tasks.

The CLIP model and the OSCAR models are two examples of such pre-trained vision-language models. Both of these models learned cross-modal representations through large-scaled pretraining. CLIP is a bi-encoder model trained through contrastive learning, and OSCAR is a transformer-based model aligning the text and image pairs. Work has been done regarding how these multimodal models can be applied to the vision-language task. CLIPscore is an example how the CLIP model can be used for a reference-free evaluation metric for image captioning.

Previous work has also been done on answering open-domain questions using text retrieval method beyond bm25 and tf-idf, such as the Dense Passenger Retrieval method introduced by Facebook AI. It has been proven to outperform traditional retrieval methods for open-domain QA tasks.

However, these retrieval methods rarely touches datasets which involves physical causality. And the retrieval model we want to build requires the system to certain level of inference here, since the effect state is not directly related to the query action language text. We are interested in learning how these retrieval methods work on datasets like this.

Please see the papers listed in the References section for the papers discussed in this section.

Methodology

In this section, I will describe what methods I plan to use to solve the problem. I plan to use a set of different IR techniques introduced in the course.

Baseline Models

Three baseline models are used in this project, including random performance, bm25, tf-idf, ifb2, lgd for the baseline performance. The bm25, tf-idf, ifb2, lgd baseline models are imported directly from the pyterrier library, while the random performance model will return a uniform distribution for relevance score in the range of 1 to 5.

The evaluation results for the baseline models are shown and discussed in the evaluation section, with the results in Figure 7. According to the results, the other baseline models are not outperforming the random performance model. This is not hard to explain, as the query has very few words in common with the most relevant documents, and models such as bm25 and tf-idf will fail to retrieve the right documents. For example, in the effect text description for the action query “someone has arranged the chairs”, the “arange” word is not even expected to be present in the effect text description.

The baseline results, though not satisfying, gives us a good motivation to work on this retrieval problem.

CLIP

CLIP is a large pre-trained multimodal model with both an image encoder and a text encoder. Here, we would like to use the pre-trained text encoder from the CLIP model to compute a similarity score between the document action language (verb + noun pairs) and the query action language. This can be easily achieved with pos-tagging packages. In our case, since we have the annotated action language for each query and document, we can directly compute the similarity score using the pre-trained CLIP text encoder and add this similarity score as a new feature. We will then use this new similarity metric for reranking the documents.

The CLIP model has already been implemented in the “transformer” python packages. I have installed this package and use their APIs for running the CLIP model. I use these APIs for computing the similarity scores for each query and document pair, the text feature similarity score and the text embedding similarity score.

Using the two similarity scores for each query and document pair, I have created three new pipelines with these features added. The first pipeline only uses the feature similarity score for reranking, the second pipeline only uses the embedding similarity score for reranking, while the third one uses both similarity scores for reranking.

ChatGPT

Improved results are seen once we incorporated the similarity scores as the doc features. However, the improvements are marginal, and this motivates us to explore new techniques to improve the retrieved result. The question we might ask ourselves is, what is the missing part of this retrieval module? What is the reason that makes it so hard for the baseline models to retrieve the correct results?

In fact, the most challenging part of this retrieval task is, the baseline models such as bm25 or tf-idf does not have any “understanding” of the causality between the action text and the effect text. To solve this problem, we need to rely on some pre-trained large language model to help us do the “reasoning” part.

Therefore, it is a natural thought to introduce the GPT-3 model into our project. The GPT-3 model is a large pre-trained generative language model with 175B parameters, and fed on enormous amount of data. If we ask the GPT-3 model, what is the effect of action, we can use this answer to expand the query so that we could obtain better retrieval results.

It probably sounds like “cheating” to use generative language models to expand the query. However, there is no reason not to use it if the pre-trained generative language model is available to us. Luckily, the free ChatGPT web API comes out on Novemember 30th gives an opportunity to try it out. I successfully set up the repo from PyChatGPT to use a Python Client to keep inquiring from the ChatGPT web API.

By using a hand-engineered query template and ask the query question to ChatGPT, we will use the answers from ChatGPT to rewrite the 140 queries in the original query dataframe by appending the answer to the query. Here is an example of the prompted result from ChatGPT for query “*Someone has arrange chairs. What could happen to the chairs?*”:

Question: Someone has arrange chairs. What could happen to the chairs?

Answer: As I mentioned in my previous response, there are many possibilities for what could happen to the chairs once they have been arranged. Some potential outcomes include:

- The chairs could be used for their intended purpose, such as providing seating for people at a meeting or event.

- The chairs could be rearranged again, either by the same person or by someone else.

- The chairs could be moved to a different location, such as being stacked and stored in a closet or shed.

- The chairs could be damaged or broken, either accidentally or intentionally.

- The chairs could be left unused for a long period of time, potentially leading to wear and tear from exposure to the elements or other factors.

There involves some data processing steps, as ChatGPT may not always answer the questions in an expected format. So, we need to extract the sentences regardless of the answer format.

Then, using the rewritten queries, we will perform experiments with the same metrics. We will also incorporate the similarity scores features for the final evaluation. See Evaluation and Results section for the final evaluation.

Evaluation and Results

In this section, we will showcase and discuss the evaluation results for the baseline models, the improved models with similarity score, the improved models with query expansion, and the improved model with both the similarity score and the query expansion. We have benmarked the models on the action-effect datasets with four metrics: mean average precision, nDCG, nDCG@5 and nDCG@10.

Baseline models

After we have preprocessed the effect text data and annotated the queries, three baseline models have been built, including random performance, bm25 and tf-idf. These models have been evaluated on four metrics: mean average precision, nDCG, nDCG@5 and nDCG@10. See Figure 7 for the evaluation results of these three baseline models.

	name	map	ndcg	nDCG@5	nDCG@10
0	random	0.048316	0.194474	0.506272	0.430963
1	bm25	0.048316	0.196708	0.515581	0.438849
2	tf-idf	0.048316	0.196708	0.515581	0.438849
3	ifb2	0.048316	0.196708	0.515581	0.438849
4	lgd	0.048316	0.196708	0.515581	0.438849

Figure 7: Evaluation results for random performance, bm25, tf-idf, ifb2, and lgd baseline models.

As we can see from the table, there is little performance difference between the random performance models and other baseline models. Therefore, this gives us a good motivation to train a better retrieval model to retrieve the most relevant effect state descriptions.

Similarity Score

We computed the similarity scores for both the text features and the text embeddings using the text encoder from the CLIP model. In this experiment, we compared the random performance model and the bm25 baseline models with three other improved models using the CLIP similarity scores, including one with feature similarity score, one with embedding similarity score, and one with both similarity scores.

See Figure 8 for the evaluation results. As we can ssee, the nDCG, nDCG@5 and nDCG@10 improve compared to the baseline models. The nDCG@5 increases significantly from 0.51 to 0.61. The feature similarity score is of less help compared to the embedding score, but the performance difference is not significant.

	name	map	ndcg	nDCG@5	nDCG@10
0	random	0.048316	0.196354	0.516988	0.437846
1	bm25	0.048316	0.196708	0.515581	0.438849
2	feat_sim	0.048316	0.216491	0.618487	0.503478
3	emb_sim	0.048316	0.217248	0.620505	0.507246
4	feat_emb_sim	0.048316	0.217248	0.620505	0.507246

Figure 8: Evaluation results for models with similarity scores added as features for reranking.

Query Expansion

For query expansion, we introduced ChatGPT as described in the previous section. Each query is expanded with the answers from ChatGPT, and the entire experiment runs with the updated queries. Note that here we need to compare across Figure 7, where we have the baseline results run on the original queries. If compare the results in Figure 9 with Figure 7, you can see that all the metrics improve significantly compared to the baseline models. The mAP goes about ten times greater than the baseline model, and the nDCG@5 goes as high as 0.9.

	name	map	ndcg	nDCG@5	nDCG@10
0	random	0.499133	0.634509	0.704358	0.693883
1	bm25	0.499133	0.678315	0.901208	0.871388
2	tf-idf	0.499133	0.678915	0.904832	0.874686
3	ifb2	0.499133	0.675936	0.893295	0.861041
4	lgd	0.499133	0.679192	0.907916	0.874060

Figure 9: Evaluation results for models with expanded queries using ChatGPT.

All-in-one

Lastly, we would like to put together everything we have so far, both the expanded queries with ChatGPT and the similarity scores computed through the text encoder from CLIP. The results are shown in Figure 10. And we can see that the performance is the best with the two features we proposed. The nDCG rises from 0.19 to 0.69, the nDCG@5 rises from 0.51 to 0.97, the nDCG@10 rises from 0.43 to 0.92 and the map rises from 0.048 to 0.49, compared to the baseline model results from Figure 7.

	name	map	ndcg	nDCG@5	nDCG@10
0	random	0.499133	0.634311	0.704115	0.688746
1	bm25	0.499133	0.678315	0.901208	0.871388
2	feat_sim	0.499133	0.694904	0.976473	0.938767
3	emb_sim	0.499133	0.692274	0.970982	0.923411
4	feat_emb_sim	0.499133	0.692274	0.970982	0.923411

Figure 10: Evaluation results for models with both expanded queries from ChatGPT and similarity scores from CLIP.

Discussion

In general, my approach does pretty well with the data that I have collected and annotated from the action-effect dataset. It successfully improves all the four metrics very significantly compared to the baseline models, including mAP, nDCG, nDCG@5, and nDCG@10.

As we can see from the experiments, we can make the following observations:

- Both the similarity scores computed through the CLIP model and the queries expanded by ChatGPT contribute to the improvement of the model performance.
- Between the two types of similarity scores computed by the CLIP model, the text embedding similarity score contributes more to the model performance than the text feature similarity score.
- Between the similarity scores features and query expansion, query expansion with ChatGPT improves the performance of the models much more significantly than the similarity scores features.
- Adding both the similarity score features and the query expansion features together leads to the most optimal performance.

Regarding the reason why my approach succeeded, I think it is because the pre-trained CLIP text encoder and the pre-trained ChatGPT is supposed to have the ability to reason between the action language and the effect texts. Unlike the traditional retrieval weighting model, where it heavily relies on the words, vocabularies, word frequency and document frequency, my approach does not rely on the index statistics heavily. Therefore, I am not surprised to see the performance gain after adding these two features to our retrieval system.

However, it remains a question whether this approach is scablabe or it can be put into practical use. Currently, my approach relies heavily on the prompted answers from ChatGPT API. The ChatGPT does require a user account and password before prompting answers from it. The Web API on the ChatGPT website also has a rate limiter, and we can not keep prompting and could have been banned for usage if we keep abusing this API. And eventaully, ChatGPT will not remain a free API and it will charge for prompting fees once it ends its beta version testing.

Conclusion

This project addresses the question of whether a retrieval system can retrieve the relevant effect text documents given an action query. We successfully implemented a retrieval system with similarity score features computed through CLIP and query expansion prompted through ChatGPT, and improved the model performance significantly compared to the baseline models.

This work proves that if we are able to identify a suitable generative language model as the reasoning backbone of a retrieval system, the retrieval system is able to make the connections between the queries and documents, and does the reasoning using the reasoning backbone. This is a step ahead compared to the traditional weighting models, such as bm25, as these models fail to detect the causality between the action query and the effect texts.

Other Things We Tried

Initially, I tried to use the GPT-3 APIs for query expansions. However, it didn't go as expected, since prompting with GPT-3 APIs is not free and even though the APIs can be easily set up, I do not have a budget for this course project.

Fortunately, the ChatGPT came out on Nov 30th. So I tried to set up a Python Client to prompt from the ChatGPT web API, and it works perfectly except for the time-to-time rate limiting.

What You Would Have Done Differently or Next

One simple experiment that can be done is to try to combine the learning-to-rank models into this pipeline as well. Such as

References

Learning Transferable Visual Models From Natural Language Supervision [Paper Link]

CLIPScore: A Reference-free Evaluation Metric for Image Captioning [Paper Link]

Oscar: Object-Semantics Aligned Pre-training for Vision-Language Tasks [Paper Link]

Dense Passage Retrieval for Open-Domain Question Answering [Paper Link]

LightningDOT: Pre-training Visual-Semantic Embeddings for Real-Time Image-Text Retrieval [Paper Link]

Image retrieval evaluation [Paper Link]

ChatGPT, https://openai.com/blog/chatgpt/

PyChatGPT, https://github.com/rawandahmad698/PyChatGPT

Transformers - CLIP, https://huggingface.co/docs/transformers/model_doc/clip

Stack Overflow, how to use CLIP text encoder
https://stackoverflow.com/questions/73593712/calculating-similarities-of-text-embeddings-using-clip