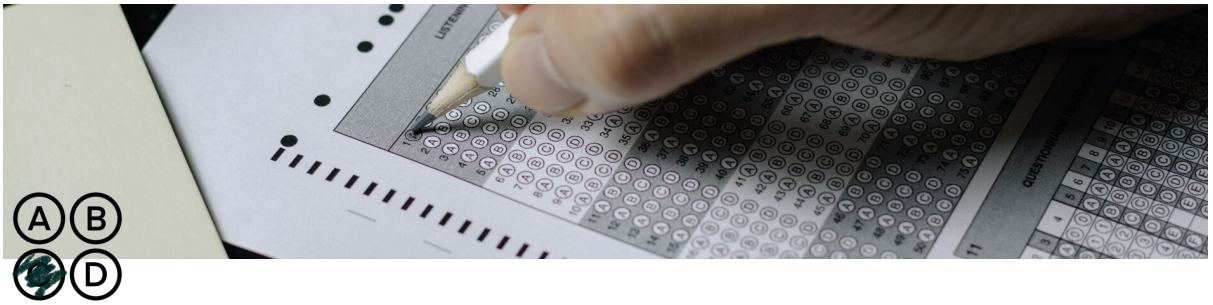

Kaggle - LLM Science Exam

Kaggle - LLM Science Exam

Use LLMs to answer difficult science questions

https://www.kaggle.com/competitions/kaggle-llm-science-exam

While working my ongoing competition - `ICR - Identifying Age-Related Conditions`, I noticed that there is another NLP competition was started in `Kaggle`.

After working on Monthly DACON - Judicial Precedent Prediction competition, I felt a bit more confident with NLP tasks.
Therefore, I decided to compete for this competition.

Here are the brief requirements of the competition.

▼ **Requirements**

Goal of the Competition

Inspired by the OpenBookQA dataset, this competition challenges participants to answer difficult science-based questions *written by a Large Language Model*.

Your work will help researchers better understand the ability of LLMs to test themselves, and the potential of LLMs that can be run in resource-constrained environments.

Context

As the scope of large language model capabilities expands, a growing area of research is using LLMs to characterize themselves. Because many preexisting NLP benchmarks have been shown to be trivial for state-of-the-art models, there

has also been interesting work showing that LLMs can be used to create more challenging tasks to test ever more powerful models.

At the same time methods like quantization and knowledge distillation are being used to effectively shrink language models and run them on more modest hardware. The Kaggle environment provides a unique lens to study this as submissions are subject to both GPU and time limits.

The dataset for this challenge was generated by giving gpt3.5 snippets of text on a range of scientific topics pulled from wikipedia, and asking it to write a multiple choice question (with a known answer), then filtering out easy questions.

Right now we estimate that the largest models run on Kaggle are around 10 billion parameters, whereas gpt3.5 clocks in at 175 billion parameters. If a question-answering model can ace a test written by a question-writing model more than 10 times its size, this would be a genuinely interesting result; on the other hand if a larger model can effectively stump a smaller one, this has compelling implications on the ability of LLMs to benchmark and test themselves.

Evaluation

Submissions are evaluated according to the Mean Average Precision @ 3 (MAP@3):

$$MAP@3 = \frac{1}{U}\sum_{u=1}^{U}\sum_{k=1}^{min(n,3)} P(k) \times rel(k)$$

where U is the number of questions in the test set, $P(k)$ is the precision at cutoff k, n is the number predictions per question, and $rel(k)$ is an indicator function equaling 1 if the item at rank k is a relevant (correct) label, zero otherwise.

Once a correct label has been scored for an individual question in the test set, that label is no longer considered relevant for that question, and additional predictions of that label are skipped in the calculation. For example, if the correct label is `A` for an observation, the following predictions all score an average precision of `1.0`.

```
[A, B, C, D, E]
[A, A, A, A, A]
[A, B, A, C, A]
```

Submission File

For each `id` in the test set, you may predict up to 3 labels for your `prediction`. The file should contain a header and have the following format:

```
id, prediction
0, A B C
1, B C A
2, C A B
etc.
```

Code Requirements

This is a Code Competition

Submissions to this competition must be made through Notebooks. In order for the "Submit" button to be active after a commit, the following conditions must be met:

- CPU Notebook <= 9 hours run-time
- GPU Notebook <= 9 hours run-time
- Internet access disabled
- Freely & publicly available external data is allowed, including pre-trained models
- Submission file must be named `submission.csv`

Please see the Code Competition FAQ for more information on how to submit. And review the code debugging doc if you are encountering submission errors.

One surprising point from the requirement was that it was allowed to add external train data.

▼ **Dataset Description**

Your challenge in this competition is to answer multiple-choice questions written by an LLM. While the specifics of the process used to generate these questions aren't public, we've included 200 sample questions with answers to show the format, and to give a general sense of the kind of questions in the test set. However, there may be a distributional shift between the sample questions and the test set, so solutions that generalize to a broad set of questions are likely to perform better. Each question consists of a `prompt` (the question), 5 options labeled `A`, `B`, `C`, `D`, and `E`, and the correct answer labeled `answer` (this holds the label of the most correct answer, as defined by the generating LLM).

This competition uses a hidden test. When your submitted notebook is scored, the actual test data (including a sample submission) will be made available to your notebook. The test set has the same format as the provided **test.csv** but has ~4000 questions that may be different is subject matter.

Files

- **train.csv** - a set of 200 questions *with* the answer column
- **test.csv** - the test set; your task it to predict the top three most probable answers given the prompt. **NOTE:** the test data you see here just a copy of the training data without the answers. The unseen re-run test set is comprised of ~4,000 different prompts.
- **sample_submission.csv** - a sample submission file in the correct format

Columns

- `prompt` - the text of the question being asked
- `A` - option A; if this option is correct, then `answer` will be `A`
- `B` - option B; if this option is correct, then `answer` will be `B`
- `C` - option C; if this option is correct, then `answer` will be `C`
- `D` - option D; if this option is correct, then `answer` will be `D`
- `E` - option E; if this option is correct, then `answer` will be `E`
- `answer` - the most correct answer, as defined by the generating LLM (one of `A`, `B`, `C`, `D`, or `E`).

To perform better on the competitions, the first thing I have done was to take a look on others' work.

After going through around 30 notebooks, I realized that many of them were using `T5` as their Main Deep Learning model.
Then, I found the research paper on T5 from `arxiv` and started going through it.

While going through the paper, I could understand how this model work and why people chose this model as their main model.

According to the paper, this model was designed to perform Text-to-Text tasks as an encoder-decoder model, unlikely to the previous model `BERT`, which was an encoder-only model.

One of the point the researchers pointed out on their paper was that using data from Wikipedia increased the performance on Question Answering tasks.

> *As a final example, using data from Wikipedia produced significant (but less dramatic) gains on SQuAD, which is a question-answering data set with passages sourced from Wikipedia.*

Moreover, they mentioned that in the previous research on BERT, pre-training the model with research papers increased performance on scientific tasks.

> *Similar observations have been made in prior work, e.g. Beltagy et al. (2019) found that pre-training BERT on text from research papers improved its performance on scientific tasks.*
> *The main lesson behind these findings is that pre-training on in-domain unlabeled data can improve performance on downstream tasks.*

Combining these two points, I decided to crawl scientific research paper, generate set of questions and answers to use as additional training data.
After a bit of research on discussion section of the competition, I realized that there are already some dataset that other people have created.
As I was unfamiliar of generating question & answers from plain text, I decided to use their dataset.

It has been few days after I last updated this log. (Sep 17 Updated)

While I was away from here, I have found DeBERTa has become popular among the competitors, and therefore I have also implemented it.

There are few versions for DeBERTa and they are :

- `deberta-base`
- `deberta-xlarge-mnli`
- `deberta-v2-xxlarge`
- `deberta-v3-small`
- `deberta-v3-base`
- `deberta-v3-large`

Among these models, I decided to go with `deberta-v3-base` and `deberta-v3-large`.
This decision was made very simply, I wanted to use the latest model.

I have first started with `deberta-v3-large` but unfortunately the notebook kept threw either `Out of Memory Error` or `Notebook Timeout Error`.

After the failure of the first attempt with `deberta-v3-large`, I tried my best to figure out why it was throwing exception.
I tried to modify all those hyperparameters, number of layers and etc.
Later found out that the model was to large that it wouldn't finish the task within 9 hours (even with the GPU).
Therefore I had to go with `deberta-v3-base` and it worked great up to max_length = 384, `batch_size = 32`, `num_sentences_include = 20` and `filter_len = 20`.

RAG

After going through others' notebooks, I realized that the point was on **RAG, Retrieval Augmented Generation**.

Based on one website that explains about RAG, LLM does not require additional background knowledge to achieve common tasks like sentiment analysis and named entity recognition.

However, LLM also need to learn the knowledge from external source to solve complex problems, like Science Exams, based on fact without hallucination.

RAG combines the information retrieval component with a text generator model. **RAG** can be fine-tuned and its internal knowledge can be modified in an efficient manner and without needing retraining of entire model.
It takes an input and retrieves a set of relevent/supporting documents given a source (e.g. Wikipedia). The documents are concatenated as context with the original input prompt and fed to the text generator which produces the final output. This makes RAG adaptive for situations where facts could evolve overtime. This is very useful as LLMs's parametric knowledge is static. RAG allows language models to bypass retraining, enabling access to the latest information for generating reliable outputs via retrieval-based generation.

Below is the overview of how the approach works.

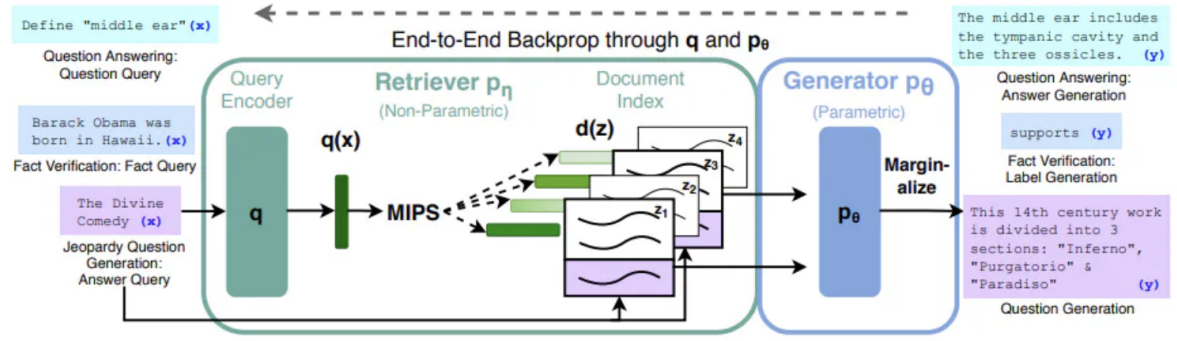

Figure 1: Overview of our approach. We combine a pre-trained retriever (*Query Encoder* + *Document Index*) with a pre-trained seq2seq model (*Generator*) and fine-tune end-to-end. For query x, we use Maximum Inner Product Search (MIPS) to find the top-K documents z_i. For final prediction y, we treat z as a latent variable and marginalize over seq2seq predictions given different documents.

LLM

The competition has ended, and below is my throwback.

These tryouts with BERT descendant models didn't make great result.
Therefore, I decided to tryout the LLM models as the competition title pointed out.

Pure LLaMa2, without any pre-training, was my first target.
It is obvious now that it wouldn't work well, however I didn't know about it well then.

I struggled learning about the model, tried to implement it for the task, it gave the score only around `0.3`. Which was worse than the `deberta` tryouts (around `0.7`).

Then, I decided to try Mistral 7B after checking a recent news that this was a brand new LLM.
This gave me a good result (`0.853`), but it I was still not enough to reach the **medal-range**.

Then I tried with `Platypus2` for the last week of the competition, which other competitions were trying with.
This gave me 0.874, the best result out of all the tryouts I have done.