question_answering

October 9, 2023

[ ]:

1 Question Answering with Generative Models on Vertex AI

Run in Colab

View on GitHub

Open in Vertex AI Workbench

1.1 Overview

Large language models can be used for various natural language processing tasks, including question-answering (Q&A). These models are trained on a vast amount text data and can generate high-quality responses to a wide range of questions. One thing to note here is that most models have cutoff dates regarding their knowledge, and asking anything too recent might yield an incomplete, imaginative or incorrect answer (i.e. a hallucination).

This notebook covers the essentials of prompts for answering questions using a generative model. In addition, it showcases the `open domain` (knowledge available on the public internet) and `closed domain` (knowledge that is more private - typically enterprise or personal knowledge).

Learn more about prompt design in the official documentation.

1.1.1 Objective

By the end of the notebook, you should be able to write prompts for the following:

- **Open domain** questions:

  - Zero-shot prompting
  - Few-shot prompting
- **Closed domain** questions:
  - Providing custom knowledge as context
  - Instruction-tune the outputs
  - Few-shot prompting

1.2 Getting Started

1.2.1 Install Vertex AI SDK

[ ]:
```
!pip install google-cloud-aiplatform --upgrade --user
```

Colab only: Uncomment the following cell to restart the kernel or use the button to restart the kernel. For Vertex AI Workbench you can restart the terminal using the button on top.

[ ]:
```
# # Automatically restart kernel after installs so that your environment can␣
↪access the new packages
# import IPython

# app = IPython.Application.instance()
# app.kernel.do_shutdown(True)
```

1.2.2 Authenticating your notebook environment

- If you are using **Colab** to run this notebook, uncomment the cell below and continue.
- If you are using **Vertex AI Workbench**, check out the setup instructions here.

[ ]:
```
# from google.colab import auth
# auth.authenticate_user()
```

1.2.3 Import libraries

Colab only: Uncomment the following cell to initialize the Vertex AI SDK. For Vertex AI Workbench, you don't need to run this.

[ ]:
```
# import vertexai

# PROJECT_ID = "[your-project-id]"  # @param {type:"string"}
# vertexai.init(project=PROJECT_ID, location="us-central1")
```

[ ]:
```
import pandas as pd
from vertexai.language_models import TextGenerationModel
```

1.2.4 Import models

[ ]:
```
generation_model = TextGenerationModel.from_pretrained("text-bison@001")
```

1.3 Question Answering

Question-answering capabilities require providing a prompt or a question that the model can use to generate a response. The prompt can be a few words or a few complete sentences, depending on the complexity of the question.

When creating a question-answering prompt, it is essential to be specific and provide as much context as possible. It helps the model understand the intent behind the question and generate a relevant response. For example, if you want to ask:

```
"What is the capital of France?",

then a good prompt could be:

"Please tell me the name of the city that serves as the capital of France."
```

In addition to being specific, the prompt should also be grammatically correct and free of spelling errors. It helps the model generate a response that is easy to understand and contains fewer errors or inaccuracies.

By providing specific, context-rich prompts, you can help the model understand the intent behind the question and generate accurate and relevant responses.

Below are some differences between the **open domain** and **closed domain** categories for question-answering prompts.

- **Open domain**: All questions whose answers are available online already. They can belong to any category, like history, geography, countries, politics, chemistry, etc. These include trivia or general knowledge questions, like:

```
Q: Who won the Olympic gold in swimming?
Q: Who is the President of [given country]?
Q: Who wrote [specific book]"?
```

Keep in mind the training cutoff of generative models, as questions involving information more recent than what the model was trained on might give incorrect or imaginative answers.

- **Closed domain**: If you have some internal knowledge base not available on the public internet, then those belong to the *closed domain* category. You can pass that "private" knowledge as context to the model. If prompted correctly, the model is more likely to answer from within the context provided and less likely to give answers beyond that from the open internet.

Consider the example of building a Q&A bot over your internal product documentation. In this case, you can pass the complete documentation to the model and prompt it only to answer based on that.

Typical prompt for **closed domain**:

```
Prompt: f""" Answer from the below context: \n\n
           context: {your knowledge base} \n
           question: {question specific to that knowledge base}  \n
           answer: {to be predicted by model} \n
        """
```

Below are some examples to understand these different types of prompts.

1.3.1 Open Domain

Zero-shot prompting

[ ]:
```
prompt = """Q: Who was President of the United States in 1955? Which party did␣
 ↪he belong to?\n
            A:
         """
print(
    generation_model.predict(
        prompt,
        max_output_tokens=256,
        temperature=0.1,
    ).text
)
```

[ ]:
```
prompt = """Q: What is the tallest mountain in the world?\n
            A:
         """
print(
    generation_model.predict(
        prompt,
        max_output_tokens=20,
        temperature=0.1,
    ).text
)
```

Few-shot prompting Let's say you want to a get a short answer from the model (like only a specific name). To do so, you can leverage a few-shot prompt and provide examples to the model to illustrate the expected behavior.

[ ]:
```
prompt = """Q: Who is the current President of France?\n
            A: Emmanuel Macron \n\n

            Q: Who invented the telephone? \n
            A: Alexander Graham Bell \n\n

            Q: Who wrote the novel "1984"?
            A: George Orwell

            Q: Who discovered penicillin?
            A:
         """
print(
    generation_model.predict(
        prompt,
        max_output_tokens=20,
        temperature=0.1,
    ).text
)
```

Zero-shot prompting vs Few-shot prompting Zero-shot prompting can be useful for quickly generating text for new tasks, but the quality of the generated text may be lower than that of a few-shot prompt with well-chosen examples. Few-shot prompting is typically better suited for tasks that require a high degree of specificity or domain-specific knowledge, but requires some additional thought and potentially data to set up the prompt.

1.3.2 Closed Domain

Adding internal knowledge as context in prompts Imagine a scenario where you would like to build a question-answering bot that takes in internal documentation and lets users ask questions about it.

In the example below, the Google Cloud Storage and content policy documentation is added to the prompt, so that the PaLM API can use that to answer subsequent questions with the provided context.

[ ]:
```
context = """
Storage and content policy \n
How durable is my data in Cloud Storage? \n
Cloud Storage is designed for 99.999999999% (11 9's) annual durability, which␣
 ↪is appropriate for even primary storage and
business-critical applications. This high durability level is achieved through␣
 ↪erasure coding that stores data pieces redundantly
across multiple devices located in multiple availability zones.
Objects written to Cloud Storage must be redundantly stored in at least two␣
 ↪different availability zones before the
write is acknowledged as successful. Checksums are stored and regularly␣
 ↪revalidated to proactively verify that the data
integrity of all data at rest as well as to detect corruption of data in␣
 ↪transit. If required, corrections are automatically
made using redundant data. Customers can optionally enable object versioning to␣
 ↪add protection against accidental deletion.
"""

question = "How is high availability achieved?"

prompt = f"""Answer the question given in the contex below:
Context: {context}?\n
Question: {question} \n
Answer:
"""

print("[Prompt]")
print(prompt)

print("[Response]")
print(
    generation_model.predict(
        prompt,
    ).text
)
```

Instruction-tuning outputs Another way to help out language models is to provide additional instructions to frame the output in the prompt. To ensure the model doesn't respond to anything outside the context, the prompt can specify that the response should be "Information not available in provided context" if that's the case.

[ ]:
```
question = "What machined are required for hosting Vertex AI models?"
prompt = f"""Answer the question given the context below as {{Context:}}. \n
If the answer is not available in the {{Context:}} and you are not confident␣
↪about the output,
please say "Information not available in provided context". \n\n
Context: {context}?\n
Question: {question} \n
Answer:
"""

print("[Prompt]")
print(prompt)

print("[Response]")
print(
    generation_model.predict(
        prompt,
        max_output_tokens=256,
        temperature=0.3,
    ).text
)
```

Few-shot prompting

[ ]:
```
prompt = """
Context:
The term "artificial intelligence" was first coined by John McCarthy in 1956.␣
↪Since then, AI has developed into a vast
field with numerous applications, ranging from self-driving cars to virtual␣
↪assistants like Siri and Alexa.

Question:
What is artificial intelligence?
```

```

Answer:
Artificial intelligence refers to the simulation of human intelligence in␣
 ↪machines that are programmed to think and learn like humans.

---

Context:
The Wright brothers, Orville and Wilbur, were two American aviation pioneers␣
 ↪who are credited with inventing and
building the world's first successful airplane and making the first controlled,␣
 ↪powered and sustained heavier-than-air human flight,
 on December 17, 1903.

Question:
Who were the Wright brothers?

Answer:
The Wright brothers were American aviation pioneers who invented and built the␣
 ↪world's first successful airplane
and made the first controlled, powered and sustained heavier-than-air human␣
 ↪flight, on December 17, 1903.

---

Context:
The Mona Lisa is a 16th-century portrait painted by Leonardo da Vinci during␣
 ↪the Italian Renaissance. It is one of
the most famous paintings in the world, known for the enigmatic smile of the␣
 ↪woman depicted in the painting.

Question:
Who painted the Mona Lisa?

Answer:

"""
print(
    generation_model.predict(
        prompt,
    ).text
)
```

1.3.3 Extractive Question-Answering

In the next example, the generative model is guided to understand the meaning of the question and the passage, and to identify the relevant information in the passage that answers the question.

The model is given a question and a passage of text, and is asked to find the answer to the question within the passage. The answer is typically a phrase or sentence.

```
[ ]: prompt = """
Background: There is evidence that there have been significant changes in␣
  ↪Amazon rainforest vegetation over the last 21,000 years through the Last␣
  ↪Glacial Maximum (LGM) and subsequent deglaciation.
Analyses of sediment deposits from Amazon basin paleo lakes and from the Amazon␣
  ↪Fan indicate that rainfall in the basin during the LGM was lower than for␣
  ↪the present, and this was almost certainly
associated with reduced moist tropical vegetation cover in the basin. There is␣
  ↪debate, however, over how extensive this reduction was. Some scientists␣
  ↪argue that the rainforest was reduced to small,
isolated refugia separated by open forest and grassland; other scientists argue␣
  ↪that the rainforest remained largely intact but extended less far to the␣
  ↪north, south, and east than is seen today.
This debate has proved difficult to resolve because the practical limitations␣
  ↪of working in the rainforest mean that data sampling is biased away from the␣
  ↪center of the Amazon basin, and both
explanations are reasonably well supported by the available data.

Q: What does LGM stands for?
A: Last Glacial Maximum.

Q: What did the analysis from the sediment deposits indicate?
A: Rainfall in the basin during the LGM was lower than for the present.

Q: What are some of scientists arguments?
A: The rainforest was reduced to small, isolated refugia separated by open␣
  ↪forest and grassland.

Q: There have been major changes in Amazon rainforest vegetation over the last␣
  ↪how many years?
A: 21,000.

Q: What caused changes in the Amazon rainforest vegetation?
A: The Last Glacial Maximum (LGM) and subsequent deglaciation

Q: What has been analyzed to compare Amazon rainfall in the past and present?
A: Sediment deposits.

Q: What has the lower rainfall in the Amazon during the LGM been attributed to?
A:
"""

print(
    generation_model.predict(
        prompt,
    ).text
)
```

1.3.4 Evaluation

You can evaluate the outputs of the question and answering task if the ground truth answers of each question are available. In zero-shot prompting, you can only use `open domain` questions. However, with `closed domain` questions, you can add context and evaluate similarly. To showcase how that will work, start by creating a simple dataframe with questions and ground truth answers.

```
[ ]: qa_data = {
         "question": [
             "In a website browser address bar, what does "www" stand for?",
             "Who was the first woman to win a Nobel Prize",
             "What is the name of the Earth's largest ocean?",
         ],
         "answer_groundtruth": ["World Wide Web", "Marie Curie", "The Pacific␣
      ↪Ocean"],
     }
     qa_data_df = pd.DataFrame(qa_data)
     qa_data_df
```

Now that you have the data with questions and ground truth answers, you can call the PaLM 2 generation model to each review row using the `apply` function. Each row will use the dynamic prompt to predict the answer using the PaLM API. We will save the results in `answer_prediction` column.

```
[ ]: def get_answer(row):
         prompt = f"""Answer the following question as precise as possible.\n\n
                 question: {row}
                 answer:
                   """
         return generation_model.predict(
             prompt=prompt,
         ).text


     qa_data_df["answer_prediction"] = qa_data_df["question"].apply(get_answer)
     qa_data_df
```

You may want to evaluate the answers predicted by the PaLM API. However, it will be more complex than the text classification since the answers may differ from ground truth and may be presented in slightly more/fewer words.

For example, you can observe the question "What is the name of the Earth's largest ocean?" and see that model predicted "Pacific Ocean" when a ground truth label is "The Pacific Ocean" with the extra "The." Now, if you use the simple classification metrics, then you will consider this as a wrong prediction since original and predicted strings have a difference. However, you can see that the answer is correct since an extra “The” is causing the issue. It’s a simple string comparison problem.

The solution to string comparison where both `ground_thruth` and `predicted` may have some extra or fewer letters, one approach is to use a fuzzy matching algorithm. Fuzzy string matching uses Levenshtein Distance to calculate the differences between two strings.

For example, the Levenshtein distance between “kitten” and “sitting” is 3, since the following 3 edits change one into the other, and there is no way to do it with fewer than 3 edits:

- kitten → sitten (substitution of “s” for “k”),
- sitten → sittin (substitution of “i” for “e”),
- sittin → sitting (insertion of “g” at the end).

Here’s another example, but this time using `fuzzywuzzy` library, which gives us the same `Levenshtein distance` between two strings but in ratio. The ratio raw score measures the string’s similarity as an int in the range [0, 100]. For two strings X and Y, the score is defined by int(round((2.0 * M / T) * 100)) where T is the total number of characters in both strings, and M is the number of matches in the two strings.

Read more here about the ratio formula :

You can see one example to understand this further.

```
String1: "this is a test"
String2: "this is a test!"

Fuzz Ratio => 97  #

Fuzz Partial Ratio => 100  #Since most characters are the same and in a similar sequence, the 
```

First, install the package `fuzzywuzzy` and `python-Levenshtein`:

```
[ ]: !pip install -q python-Levenshtein --upgrade --user
     !pip install -q fuzzywuzzy --upgrade --user
```

Then compute a score to perform fuzzy matching:

```
[ ]: from fuzzywuzzy import fuzz


     def get_fuzzy_match(df):
         return fuzz.partial_ratio(df["answer_groundtruth"], df["answer_prediction"])


     qa_data_df["match_score"] = qa_data_df.apply(get_fuzzy_match, axis=1)
     qa_data_df
```

Now that you have the individual match score (partial), you can take the mean or average of the whole column to get a sense of overall data. Scores closer to 100 mean PaLM 2 can predict closer to ground truth; if the score is towards 50 or 0, it did not perform well.

[ ]:
```
print(
    "the average match score of all predicted answer from PaLM 2 is : ",
    qa_data_df["match_score"].mean(),
    " %",
)
```

In this case, you get 100% as the mean score, even though some predictions were missing some words. That means you are very close to the ground truth, and some answers are just missing the exact verboseness of the ground truth.